Joonas Rapila

PowerShell skriptiohjelmoinnissa

NETCONF-ohjelmistokirjaston prototyyppi

Metropolia Ammattikorkeakoulu
Tietoverkot
Tietotekniikka
Opinnäytetyö
21.5.2012

Metropolia

Tiivistelmä

Tekijä(t) Otsikko Sivumäärä Aika	Joonas Rapila Powershell skriptiohjelmoinnissa - NETCONF-ohjelmistokirjaston prototyyppi 43 sivua + 1 liitettä 21.5.2012
Tutkinto	Tietotekniikan (AMK) Insinööri
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Tietoverkot
Ohjaaja(t)	Yliopettaja Salonen Janne

Opinnäytetyössä tutkittiin Microsoft Powershell -ympäristön skriptikielen soveltuvuutta ohjelmoinnissa. Työn käytännön osuudessa ohjelmoitiin prototyyppi NETCONF-protokollan toteuttavasta ohjelmistokirjastosta.

Työn yhtenä tarkoituksena oli selvittää, onnistuuko ohjelmakirjaston prototyypin toteutus siten, että käytetään ainoastaan Powershell-ympäristön skriptikieltä. Toisena tarkoituksena oli selvittää NETCONF-protokollan kypsyys verkkolaitteen konfiguraation hallinnan työkaluna.

Opinnäytetyön raportointiosio alkaa esittelemällä Powershell-ympäristön keskeisimmät piirteet. Siitä jatketaan esittelemällä NETCONF-protokolla sekä protokollan määrittelevässä standardissa RFC 4741 esitetyn pakollisen kuljetusprotokollan Secure Shell -protokollan toiminta NETCONF-istunnon muodostuksessa. Lopuksi raportissa esitellään käytännön osiossa tehty ohjelmistokirjasto.

Työ suoritettiin VMware Workstation -ohjelmaan asennetuilla virtuaalikoneilla. NETCONF-protokollan palvelimena käytettiin FreeBSD-käyttöjärjestelmän päälle asennettua Juniper Networksin Junos-käyttöjärjestelmää Olive-emulaattorilla.

Ohjelmointityö suoritettiin Microsoftin Powershell.exe-konsolipohjaisella sovelluksella sekä graafisella Powershell ISE -ohjelmalla.

Opinnäytetyössä päädyttiin siihen tulokseen, että Powershell-ympäristö soveltuu prototyyppien ja ylläpidollisten skriptien alustaksi. Siinä todettiin kuitenkin myös se, että Powershell-ympäristölle löytyvä dokumentaatio ei vielä ole niin laaja kuin voisi toivoa. Lisäksi erilaisten ohjelmointia tukevien työkalujen, kuten ohjelmointiympäristöjen saatavuus on, toistaiseksi vielä melko suppea. Tästä syystä laajempien projektien ratkaisuna jokin muu ohjelmointialusta on vielä toistaiseksi parempi ratkaisu.

Avainsanat	Powershell, NETCONF, Skriptiohjelmointi, Komentosarja, Komentorivi

Metropolia

Abstract

Author(s) Title Number of Pages Date	Joonas Rapila The use of Powershell in script-based application development – A prototype of a NETCONF-based programming library. 43 pages + 1 appendices 21 May 2012
Degree	Bachelor of Engineering
Degree Programme	Computer Science
Specialisation option	Computer Networks
Instructor(s)	Janne Salonen, Principal Lecturer

This thesis researched the use of the Microsoft Powershell scripting language as a solution for programming. It was used for creating a prototype of a programming library for the NETCONF configuration management protocol.

The main objective of the thesis was to find out whether the Powershell scripting language would qualify as a primary solution to be used for programming. Also as a side aspect the maturity of the NETCONF protocol was being researched, by using a NETCONF server based on the Juniper Networks Junos operating system, which was being run as a virtual machine on the FreeBSD-based Olive emulator.

First the thesis researches into the core features of the Microsoft Powershell technology. From there it delves into the main parts of the NETCONF configuration protocol. It represents in detail how the NETCONF protocol is to be used with an SSH connection, the only mandatory transport protocol defined for the NETCONF protocol in RFC 4741.

Finally the program library and its use are represented. The thesis concludes that although the Powershell scripting language is suited for a scripting solution for prototypes and administrative scripts, it is still quite new and does not offer quite as good documentation and development tools as some other similar scripting languages do.

Keywords	Powershell, NETCONF, Scripting

Sisällys

Lyhenteet

CMDLET	.NET-sovelluskehyksellä käännetty Powerhell-teknologian Cmdlet-luokkaan perustuva sovellus.
NETCONF	Verkkolaitteiden konfiguraatioiden hallintaan kehitetty protokolla.
.NET	Microsoftin kehittämä sovelluskehys, joka sisältää ajonaikaisen ympäristön sekä luokkakirjastot.
PowerShell	Automaatioympäristö tietojärjestelmien ylläpitoon.
Powershell.exe	Powershell-isäntäsovelluksen toteuttava konsolipohjainen komentotulkki.
Powershell ISE	*Powershell Integrated Script Environment*. Powershell-isäntäsovelluksen toteuttava graafinen komentotulkki ja skriptieditori.
RPC	*Remote Procedure Call*. Etäkutsutekniikka, jonka avulla voidaan ohjelmallisesti kutsua ja käynnistää prosesseja verkon yli toisilla laitteilla.
Skriptikieli	Korkean tason dynaamisesti tyypitetty ohjelmointikieli.
XML	*Extensive Markup Language*. Metakieli, jota käytetään kuvaamaan tietoa tiedosta. Määrittelee käytetyn datan formaatin ja sallitut arvot.

1 Johdanto

Opinnäytetyössä tutkittiin verkkolaitteiden konfiguraatioiden hallintaan kehitettyä NETCONF-protokollaa sekä Microsoftin Powershell-komentorivin ja komentosarjakielen ympäristöä. Siinä toteutettiin Powershell-skriptikielellä NETCONF-protokollaan pohjautuva sovellus konfiguraatioiden hallintaan.

Opinnäytetyön tarkoituksena oli selvittää, voidaanko Powershell-ympäristöllä kehittää ohjelmisto, joka pohjautuu pääasiassa Powershell-ympäristön toteuttamaan komentosarjakieleen.

NETCONF on RPC-arkkitehtuuriin pohjautuva XML-koodattu protokolla, joka määrittelee rajatun määrän toimenpiteitä konfiguraatioiden hallintaan. Se kehitettiin korvaamaan sekä joissain tapauksissa täydentämään olemassa olevien ratkaisujen puutteitta liittyen varmennettuun transaktiopohjaiseen yhden tai useamman laitteen hallintaan.

Opinnäytetyössä tehdyn ohjelman tarkoituksena ei ollut toteuttaa kaikkia NETCONF-protokollalle määriteltyjä ominaisuuksia, vaan ohjelmoida minimivaatimukset täyttävä ratkaisu, joka mahdollisti Powershell-ympäristön olennaisimpien ominaisuuksien demonstraation alustana ohjelmistopohjaiselle kehitykselle.

Opinnäytetyön raporttiosio etenee siten, että toisessa luvussa käydään läpi Powershell ympäristön tärkeimmät piirteet. Kolmannessa luvussa esitellään NETCONF-protokollan komponentit ja toiminta. Neljännessä luvussa tarkastellaan toteutettua ohjelmaa. Viidennessä luvussa pohditaan työn tuloksia.

2 Powershell-ympäristö

2.1 Powershell-kehityksen taustat

Microsoft on perinteisesti nojannut kehityksessään graafisiin käyttöliittymiin eikä ole juurikaan kehittänyt komentoriviin pohjautuvaa hallintaliittymää. Tämän voidaan katsoa johtuvan pääasiassa kahdesta syystä. Ensinnäkin kohderyhmä Microsoftin kehittämälle Windows-käyttöjärjestelmälle on ollut graafista käyttöliittymää suosivat käyttäjät. Toinen syy on se, että graafisia käyttöliittymiä käyttävät ohjelmat hyödyntävät binäärisiä ohjelmointirajapintoja keskustellessaan järjestelmälle, ja nämä ovat lähes poikkeuksetta epäyhteensopivia tai hankalasti toteutettavissa komentoriviä hyödyntävistä ohjelmista.

Microsoft käynnisti 2000-luvun alussa tutkimuksen, jossa selvitettiin kehitystä vaativia osa-alueita palvelimille suunnatussa tarjonnassaan. Tutkimuksen tuloksessa todettiin, että Windows-pohjaisten käyttöjärjestelmien ja niissä suoritettavien ohjelmien ylläpidolle (etenkin komentorivi ja skriptaus) kehitetyt ratkaisut olivat aliedustettuina ja vaativat aktiivista kehittämistä. [1, s. 7.]

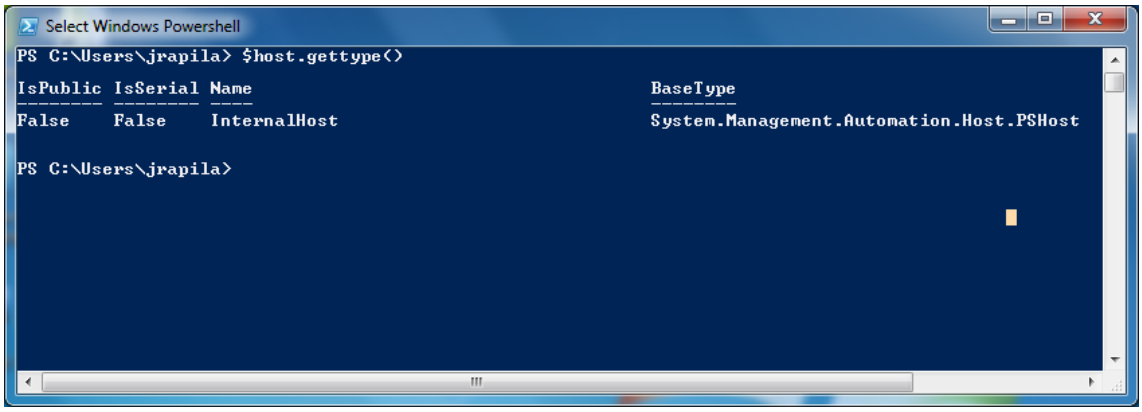
Vuonna 2002 Microsoftilla työskentelevä Powershell-ympäristön keksijä Jeffrey Snover julkaisi manifestin, jossa hän kuvaili Monadiksi nimeämäänsä ratkaisua Windows-ympäristölle suunnatuksi automaatioalustaksi. Manifestin alussa Snover esitteli olemassa olevien toteutusten ongelmia. Snover [2] totesi manifestissaan, että vaikka Windows-ympäristön järjestelmille on kehitetty rajapintoja ja oliomalleja usealle ohjelmointikielelle, ratkaisut oli suunnattu lähinnä järjestelmäohjelmoijille.

Vuonna 2005 Microsoft julkaisi ensimmäisen julkisen beetaversion Monadista, ja vuonna 2006 julkaistiin ensimmäinen versio ohjelmasta, jolle oli annettu nimi Powershell. Toinen versio, Powershell versio 2, julkaistiin vuonna 2009. Vuonna 2012 Microsoft julkaisi beetajulkaisun Powershell-ympäristön versiosta 3. [3.]

2.2 Powershell-ympäristön sisäinen toiminta

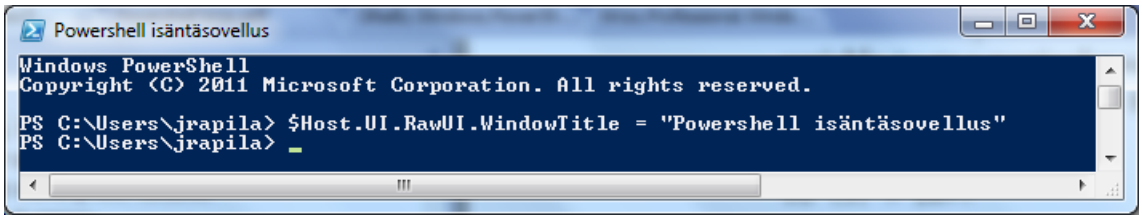
Powershell-ympäristö koostuu kahdesta ydinosasta: Powershell-moottorista (Powershell engine), joka tulkitsee komennot, ja isäntäsovelluksesta (host application), joka toimii käyttöliittymänä käyttäjän ja moottorin välillä [1, s. 14].

Isäntäsovellus voi olla konsoli-, Windows- tai web-pohjainen sovellus [4]. Windows-käyttöjärjestelmän mukana toimitetaan oletuksena kaksi isäntäsovellusta: konsolipohjainen Powershell.exe ja graafinen Powershell ISE. Käyttäjä voi tarvittaessa luoda myös oman isäntäsovelluksen .NET-sovelluskehyksen System.Management.Automation.Host-nimiavaruudesta löytyvällä PSHost-luokalla. Kuten kuviosta 1 nähdään, periytyy isäntäsovellus Powershell.exe PSHost-luokasta.



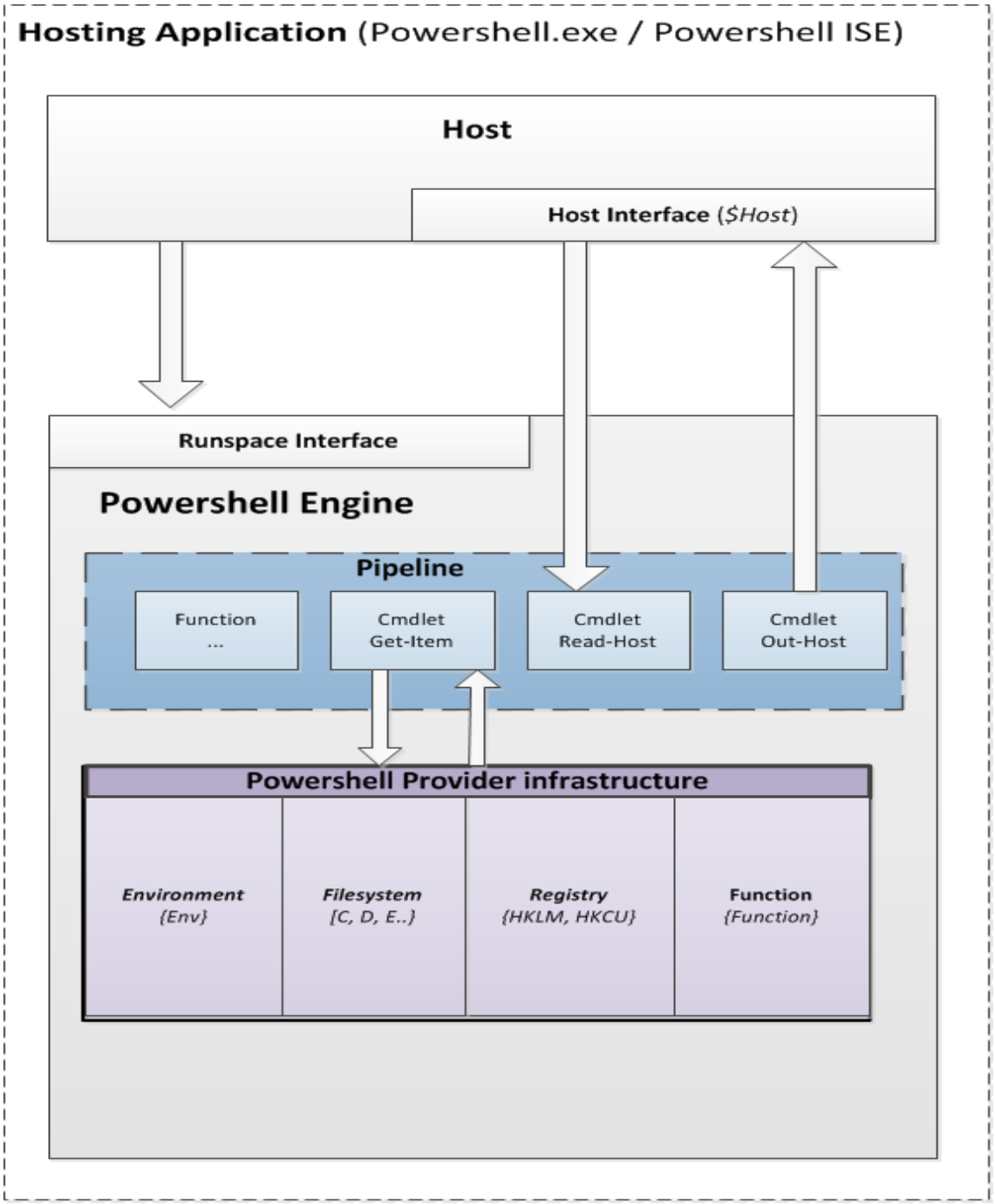
Kuvio 1. Powershell.exe-isäntäsovellus periytyy .NET-sovelluskehyksen PSHost-luokasta.

Kuviossa 1 on esitetty $host-muuttuja oli sisäänrakennettu muuttuja, joka löytyy kaikista PSHost-luokkaa käyttävistä tai siitä periytyvistä isäntäsovelluksista. $Host-muuttujasta pääsi käsiksi isäntäsovellukseen ja sen avulla voitiin esimerkiksi vaikuttaa isäntäsovelluksen ulkonäköön liittyvin asetuksiin niin kuin kuviossa 2 on tehty. Kuviossa 2 vaihdettiin ikkunan nimi ”Windows Powershell” -nimestä ”Powershell Isäntäsovellus” -nimeen.



Kuvio 2. $Host-muuttuja mahdollisti pääsyn isäntäsovellukseen.

Kuviossa 3 esitettiin Powershell-ympäristön ydinkomponentit ja toiminta. Kuviossa isäntäsovellus (host application) isännöi Powershell-moottoria (engine) ja varsinaista isäntäsovellusta (host). Tässä haluttiin kuvata sitä, että molemmat toimivat samassa prosessissa. Microsoftin Windows-käyttöjärjestelmän mukana toimitettavissa isäntäsovelluksissa, Powershell.exe ja Powershell ISE, Powershell-moottori ja isäntäsovellus kuuluvat samaan prosessiin. On kuitenkin mahdollista suorittaa moottori ja isäntäsovellus erillisissä prosesseissa. Tämä tosin vaatii räätälöidyn isäntäsovelluksen toteuttamisen PSHost-luokasta [5].

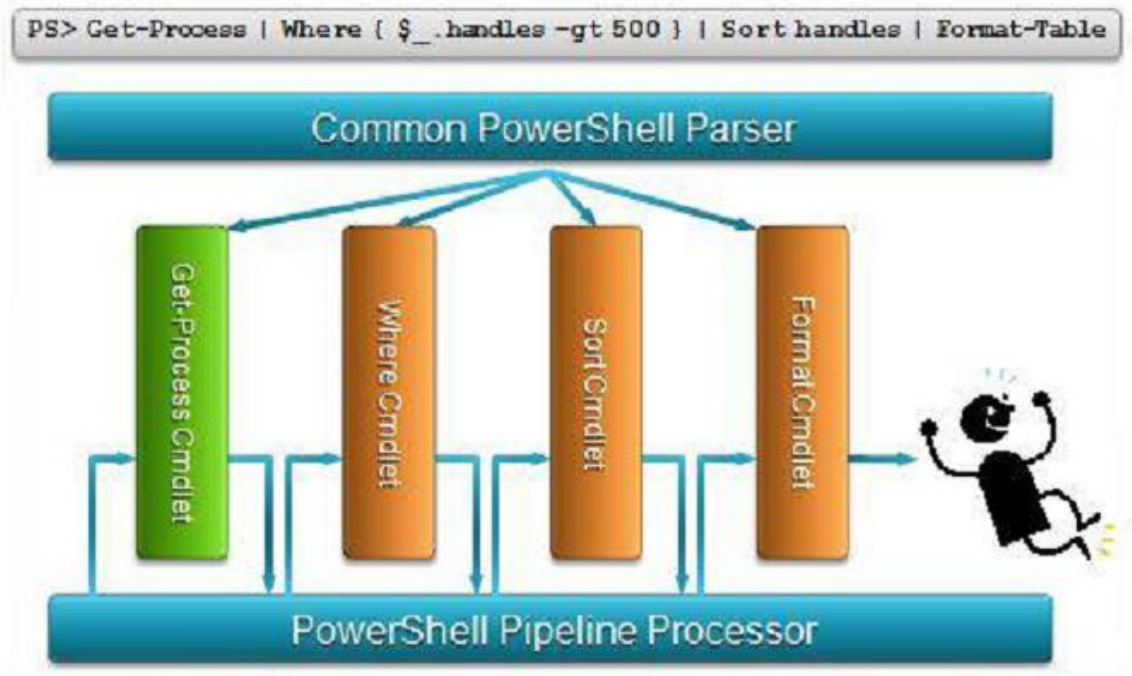


Kuvio 3. Powershell-ympäristön sisäinen toiminta ja komponentit. Kuva on muokattu Microsoftin MSDN-dokumenttikirjastossa löytyvän kuvan [4] pohjalta.

Kuviossa 3 Isäntäsovellus (host) kutsui moottoria, josta käytetään myös nimitystä Powershell runtime [4], runspace-liittymän kautta (runspace interface). Powershell-moottori toteuttaa suoritusympäristön skripti-, cmdlet-, funktio-, provider- ja ulkoisesti suoritettaville ohjelmille (kuten vaikka ohjelma ping.exe).

Runspace-olio, joka on abstraktio Powershell-moottorista, vastaa pitkälti samaa asiaa kuin Powershell-istunto (jota kuvaa PSsession-olio) [1, s. 619]. Runspace-olio luodaan kutsumalla isäntäsovelluksen Main()-funktiosta RunspaceFactory-luokan CreateRunspace-metodia [4].

Runspace-olioon luodaan yksi tai useampi putkijono (pipeline), jossa komennot suoritetaan. Kaikki Powershellin käsittelemä data on .NET-sovelluskehyksen ohjelmistokirjastojen mukaisia olioita. Kuviossa 4 on esitelty putkijonon toiminta.



Kuvio 4. Esimerkki Powershellissä tapahtuvasta komentojen putkituksesta [1, s. 61].

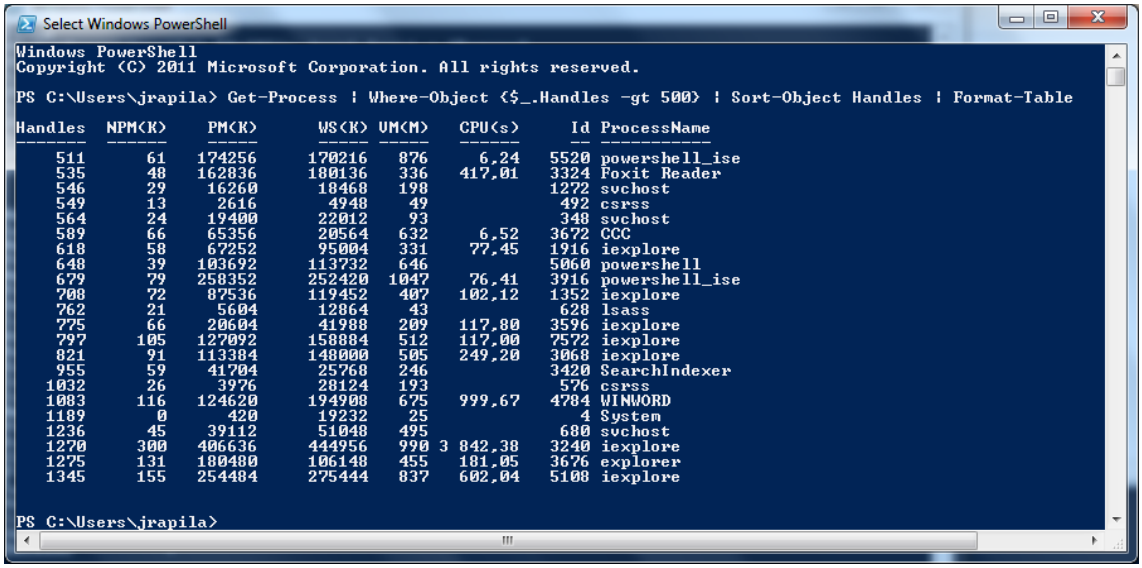
Kuviossa 4 harmaalla taustalla merkitty ikkuna sisältää putkijonon, jossa on neljä komentoa. Aluksi putkijono syötetään Powershell-tulkille (Common Powershell Parser), joka selvittää, mitä komennot ovat, miten ne tulisi suorittaa sekä miten niille syötetyt parametrit ja parametrien arvot käsitellään. Käsiteltyään komennot tulkki palauttaa komennot putkijonolle putkijonon hyväksymässä muodossa, joka sitten suorittaa komennot seuraavassa luvussa esitettyjen toimenpiteiden mukaisesti. [1, s. 61.]

Putkijonot (pipeline)

Suorittaminen putkijonossa koostuu kolmesta vaiheesta, jotka ovat *begin*, *process* ja *end*. Jokainen vaihe suoritetaan kaikille komennoille järjestyksessä ensimmäisestä viimeiseen, ennen kuin siirrytään seuraavaan vaiheeseen. Komennon toteuttaja (ohjelmoija) määrittelee kunkin vaiheen sisältämät operaatiot. Kaikkia vaiheita ei tarvitse toteuttaa, ja usein käytetäänkin pelkästään process-vaihetta.

Aluksi suoritetaan begin-vaihe. Begin-vaiheessa suoritetaan määritelty operaatio kaikille putkijonossa sijaitseville olioille. Begin-vaihetta saatetaan käyttää vaikka alustamaan muuttujia. Begin-vaiheen jälkeen siirrytään process-vaiheeseen. Jos ensimmäisen komennon process-lausekkeesta syntyy tulostettavia olioita, lähetetään oliot seuraavalle komennolle yksi kerrallaan sitä mukaa, kun tulosteita syntyy (sama kaava toistuu jokaisessa putkijonon komennossa). Kun putkijonon viimeinen komento on käsitellyt olion, lähetetään olio isäntäprosessille, jonka tehtäväksi jää olion käsittely (esimerkiksi tiettyjen arvojen tulostus käsiteltävästä oliosta käyttäjälle). Kun process-lauseke on suoritettu, suoritetaan end-lauseke, jonka suoritus tapahtuu samoin kuin begin-lausekkeen. [1, s. 61.]

Kuvion 5 putkijonon get-process-komento tuottaa .NET-ohjelmistokehyksen System.Diagnostics.Process-luokan olioita.



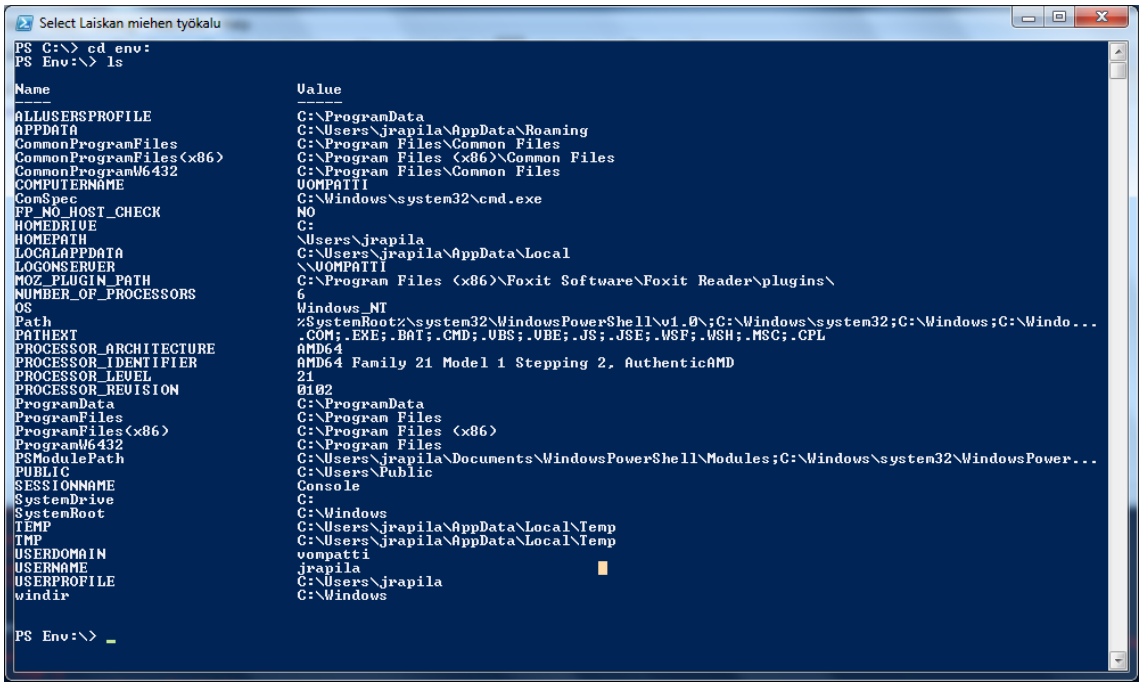
Kuvio 5. get-process-komennon käsittely putkijonossa.

Sitä mukaa kun putkijonossa sijaitsevat komennot käsittelevät get-process-komennon palauttamia olioita (process-vaiheessa) lähetetään, ne isäntäsovellukselle.

Isäntäsovellus, joka on yleensä joko Powershell.exe tai sitten Powershell ISE, käsittelee putkijonossa olevan tiedon synkronisesti First In First Out (FIFO) -periaatetta noudattaen. On myös mahdollista käsitellä putkijonojen dataa asynkronisesti toteuttamalla räätälöidyn isäntäsovelluksen. [4; 6, s. 9-10.]

Provider-liitännäinen

Kuviossa 3 on myös esitetty Provider-niminen komponentti. Provider-liitännäiset ovat .NET-sovelluskehykseen pohjautuvia ohjelmia, joiden avulla on mahdollista käsitellä erilaisissa tietovarastoissa sijaitsevaa tietoa samaan tapaan kuin esimerkiksi tiedostojärjestelmiä ollaan totuttu käsittelemään. Microsoft toimittaa Powershell-version 2 mukana Provider-komponentit muun muassa rekisterien, tiedostojärjestelmien, ympäristömuuttujien ja sertifikaattien käsittelyyn. Kuviossa 6 esitellään ympäristömuuttujien tarkastelu Provider-komponentin välityksellä.



Kuvio 6. Windows-ympäristömuuttujien tarkastelu Provider-komponentista.

2.3 Komentotyypit

Isäntäsovelluksesta voidaan kutsua komentoja, joita ovat cmdletit, funktiot, skriptit ja Powershell-ympäristön ulkopuoliset komennot kuten konsolisovellukset ping.exe tai ipconfig.exe. Cmdletit, funktiot ja skriptit ovat osa Powershell-ympäristöä ja tukevat täten putkijonossa käytettäviä begin-, process- ja end-lausekkeita.

Cmdlet

Cmdletit ovat .NET-sovelluskehykseen pohjautuvia komentoja, jotka periytyvät cmdlet-luokasta. Cmdlet-ohjelmat käännetään dynaamisiksi linkkikirjastoiksi ja ladataan isäntäsovelluksen prosessiin prosessin käynnistyessä. Microsoft on luonut satoja cmdlet-ohjelmia Powershell-ympäristölle. Cmdlet-ohjelmien nimeäminen on tarkkaan määritelty. Ne tulee nimetä verbi-substantiivi-nimeämiskäytäntöä seuraten, missä verbi kuvaa tekemistä ja substantiivi käsiteltävää oliota. Esimerkiksi prosessit noutavan cmdlet-ohjelman nimi on get-process tai prosessin pysäyttävä ohjelma on stop-process.

Kuviossa 7 on esimerkki C#-kielellä kirjoitetun cmdlet-ohjelman lähdekoodista. Ohjelma tulostaa sille syötetyn datan. Cmdlet-ohjelma perii ominaisuudet cmdlet-luokalta. Cmdlet-ohjelmassa parametrit määritellään erityisellä parameter-attribuutilla. Parameter-attribuutille voidaan antaa useita arvoja ja sen avulla voidaan esimerkiksi määritellä pakolliset parametrit, sallitut arvot tai se, tukeeko cmdlet-ohjelma putkijonoja (kuvassa ValueFromPipeLine-arvo). Powershell-moottorin parseri vastaa cmdlet-ohjelmalle annettujen syötteiden ja parametrien validiuden tarkastuksesta. Kehittyneiden funktioiden syntaksi muistuttaa hyvin läheisesti cmdlet-ohjelmien syntaksia ja niissä on muun muassa tuki samoille parametrien parseroinneille Powershell-moottorin toimesta.

```
[Cmdlet("Write", "InputObject")]
public class MyWriteInputObjectCmdlet : Cmdlet
{
    [Parameter]
    public string Parameter1;

    [Parameter(Mandatory = true, ValueFromPipeline=true)]
    public string InputObject;

    protected override void ProcessRecord()
    {
        if (Parameter1 != null)
                WriteObject(Parameter1 +  ":" +  InputObject);
            else
                WriteObject(InputObject);
    }
}
```

Class declaration

Marks parameter

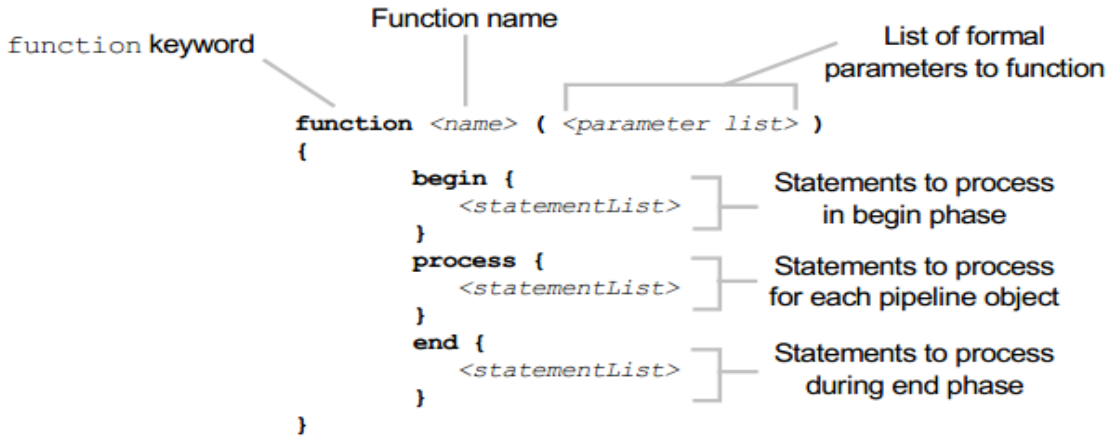
Takes pipeline input

Process block

Kuvio 7. Esimerkki C#-kielellä kirjoitetun cmdlet-ohjelman rungosta.

Funktiot

Funktiot ovat skripteissä tai isäntäsovelluksen istunnosta käytettäviä ohjelmapätkiä. Ne sijaitsevat muistissa suorituksen ajan, ja ne tuhotaan suorituksen loputtua. Kuviossa 8 on kuvattuna funktion rakenne. Funktio määritellään function-sanalla, jonka perään tulee funktion nimi ja lopuksi parametrit sulkuihin. Funktiolla ei välttämättä tarvitse olla parametrejä. Funktioita voidaan hallita Provider-komponentilla, josta voidaan muun muassa listata istunnossa määritellyt funktiot tai poistaa niitä. Funktiot ovat aktiivisia ainoastaan istunnon ajan, jos niitä ei tallenneta tiedostoihin. Kuviossa on esitelty begin-, process- ja end-vaiheet, joiden toiminta selitettiin putkijonojen yhteydessä.



Kuvio 8. Funktion rakenne ja begin-, process- ja end-vaiheet.

Powershellin versiossa 2 julkaistiin tuki niin kutsutuille kehittyneille funktioille (advanced functions), jotka tuovat cmdlet-ohjelmien kaltaiset ominaisuudet funktioille. Niiden rakenne eroaa normaaleista funktiosta ja muistuttaa enemmän cmdlet-ohjelmien rakennetta. Kehittyneillä funktioilla voidaan tehdä lähes samat asiat kuin cmdlet-ohjelmilla, joskaan ei aivan kaikkia.

Skriptit

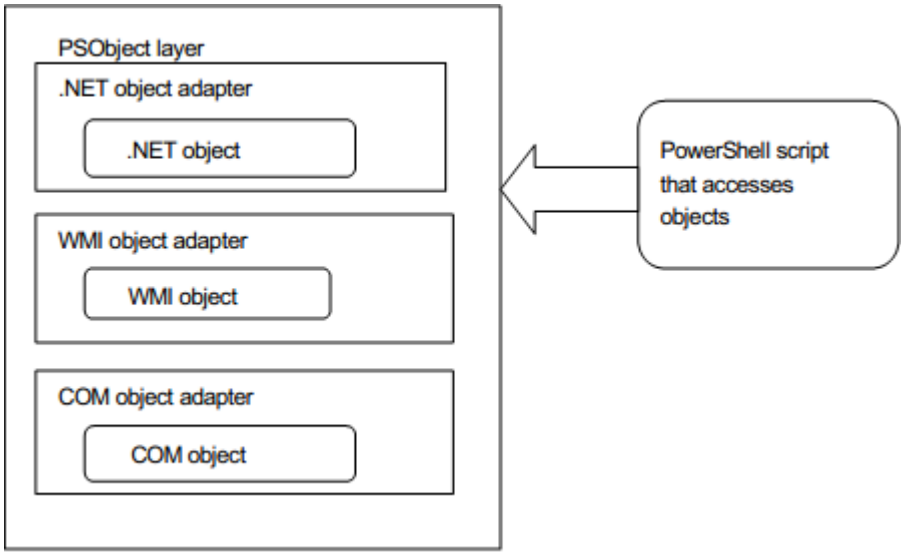
Skriptikomennot ovat .ps1-päätteisissä tekstitiedostoissa sijaitsevia komentoja. Skriptikomennot koostuvat yhdestä tai useammasta funktiosta.

Ulkoiset komennot

Ulkoisilla komennoilla tarkoitetaan Powershell-ympäristöön kuulumattomia ohjelmia. Ne voivat olla esimerkiksi konsolipohjaisia ohjelmia, kuten ping.exe tai netsh.exe. Kaikki isäntäsovellukset eivät välttämättä tue kaikkien ulkoisten komentojen suorittamista. Esimerkiksi Powershell ISE ei tue joitain konsolipohjaisia sovelluksia, kuten edit.com, diskpart.exe tai powershell.exe. Powershell.exe on siis ulkoinen konsoliohjelma, joka on itse asiassa kirjoitettu C++-ohjelmointikielellä. Powershell.exe isännöi .NET CLR-moottoria (Common Language Runtime), josta sillä on pääsy .NET-sovelluskehyksen kirjastoihin. [7.]

2.4 Powershell Extended Type System (ETS)

Powershell on rakennettu .NET-sovelluskehyksen päälle, ja se käyttää .NET-luokkia tietotyyppiensä määrittelyyn. Kuitenkin, koska .NET on kehitetty kääntämistä tukeville ohjelmointikielille ja suunnattu ohjelmoijille, ei se sellaisenaan sovellu Powershell-ympäristössä suoritettaville skripti-ohjelmille. Tämän puutteen paikkaamiseksi Powershell-ympäristön kehittäjät laajensivat .NET-sovelluskehystä niin kutsutulla Powershell Extended Type System (ETS) -kirjastolla. ETS muodostaa rungon Powershell-ympäristön tietotyypeille. ETS määrittelee PSObject-luokan, johon kaikki Powershellissä luotujen olioiden ilmentymät perustuvat. Kuviossa 9 on esitetty miten, PSObject-luokan olio sinetöi muun tyyppiset oliot sisäänsä.



Kuvio 9. PSObject-luokan olio toimii säiliönä kaikille muille olioille Powershell-ympäristössä [1, s. 76].

Lisäksi ETS mahdollistaa Powershellin tukemien tietotyyppien laajentamisen mahdollistaen omien muuttujien, metodien ja nimiavaruuksien lisäämisen olemassa oleviin tai uusiin tietotyyppeihin. [6, s. 29.]

3 NETCONF

3.1 Esittely

NETCONF-protokolla on tietoverkkojen hallintaan kehitetty protokolla. Sen avulla voidaan hallita yhdestä tai useasta laitteesta koostuvia tietoverkkoja. Sitä voidaan hyödyntää, kun halutaan ottaa jokin palvelu tietoverkossa käyttöön. Tietoverkkoja hallittaessa voitaisiin esimerkiksi asentaa uusi Virtual Private Network (VPN) -verkko, joka vaatii asetusten tekemistä useampaan laitteeseen. Operaation onnistuminen vaatii mekanismin, jolla voidaan määrittää asetukset yhteen tai useampaan laitteeseen sekä varmistaa operaation onnistuminen. NETCONF mahdollistaa tämän kaltaiset toimenpiteet tarjoamalla standardoidun ohjelmointirajapinnan asetusten hallitulle määrittämiselle. [8, s. 105.]

NETCONF-protokolla sai alkunsa kesäkuussa 2002 verkon hallintaa käsittelevässä työpajassa, jonka Internet Architecture Board (IAB) oli järjestänyt verkko- sekä palve-

luoperaattoreiden ja standardointia käsittelevien organisaatioiden pyynnöstä. Työpaja kutsuttiin koolle, koska useat operaattorit ja protokollien kehitykseen osallistuneet henkilöt kokivat nykyiset verkon ja asetusten hallintaan kehitetyt työkalut riittämättömiksi. Aiemmin kehitettyjä ratkaisuja ovat esimerkiksi Simple Network Management Protocol (SNMP) ja Common Open Policy Service (COPS) -protokollat verkonhallintaan ja Core Information Model (CIM) sekä Management Information Base (MIB) -tietomallit tiedon standardinmukaiselle esittämiselle. RFC 3535 -dokumentti käsittelee kyseistä työpajaa ja sen tuloksia. Työpajan päätöslauselmassa todettiin, että sen hetkiset ratkaisut olivat riittämättömiä ja Internet Engineering Task Force (IETF) -järjestölle annettiin tehtäväksi kehittää uusi mekanismi korjaamaan puutteita. Ratkaisulle annettiin muun muassa seuraavat kriteerit:

- Ratkaisun tulisi olla helppo ottaa käyttöön ja hallita.
- Sen tulisi selkeästi erottaa toisistaan palvelimen asetusten ja tilan esittämisen.
- Mahdollistaa verkon tarkastelu kokonaisuutena yksittäisten laitteiden sijaan, asetusten määrittämisen perspektiivistä. [8, s. 105-107.]

Työpajan seurauksena IETF käynnisti NETCONF-hankkeen. Hanketta toteuttavalla ryhmällä oli mandaattina kehittää ratkaisu, joka tukisi esimerkiksi seuraavia tavoitteita:

- Ratkaisun tulisi tukea laajennettavuutta siinä määrin, että laitevalmistajat mahdollistavat pääsyn kaikkiin asetuksiin hyödyntäen yhtä protokollaa.
- Ratkaisun tulisi tarjota ohjelmistorajapinta, jolla voitaisiin välttyä ruutukaappaukselta sekä formaattiin liittyviltä ongelmilta eri palvelimen ohjelmistoversioiden kesken.
- Ratkaisun tulisi olla tekstipohjainen niin, että tietoon päästään käsiksi ilman erillisiä työkaluja.
- Mekanismin tulisi olla yhteensopiva aiempien ratkaisujen kanssa, kuten esimerkiksi tunnistusta ja asetustietokantoja hyödyntävien standardoitujen ratkaisujen kanssa.
- Mahdollistaa palvelujen asetusten määritykseen liittyvät transaktiot, tukemalla ominaisuuksia kuten tiedostojen lukitsemista ja palauttamista operaatiota edeltävään tilaan.
- Toimittaa mekanismi ilmoitusten välittämiseen.

- Määrittää syntaksi ja protokolla ilmoituksille sekä ilmoitusten välittämiselle. [9.]

IETF-työryhmän tuloksena julkaistiin joulukuussa 2006 RFC 4741: Network Configuration Protocol. Päivitetty määritys, RFC 6241: Network Configuration Protocol, julkaistiin kesäkuussa 2011. Lisäksi työryhmä on julkaissut useita muita NETCONF-protokollaan liittyviä määrityksiä, jotka löytyvät IETF:n ylläpitämiltä WWW-sivuilta [10]. Koska NETCONF-protokollan toiseen versioon, jonka määrittelee RFC 6241, pohjautuvia ratkaisuja ei ole juurikaan käytössä, päätettiin opinnäytetyössä käyttää pohjana aiempaa RFC 4741 -dokumenttia.

XML-merkinnät NETCONF-protokollassa

NETCONF-protkollassa kaikki tieto välitetään XML-koodattuna. Käsitellyltä tiedolta vaaditaan, että se on hyvin muotoiltua (well-formed) sekä UTF-8-merkistön mukaista. Jos palvelin vastaanottaa viestin, joka ei ole joko hyvin muotoiltu tai UTF-8-merkistön mukainen, sen tulee pyrkiä palauttamaan virheilmoitus, tai virheilmoituksen epäonnistuessa palvelimen on katkaistava istunto. NETCONF-protokollan kaikki elementit sisältyvät XML-nimiavaruuteen urn:ietf:params:xml:ns:netconf:base:1.0. Nimiavaruus sisällytetään attribuuttina *kuljetuskerroksen* XML-koodatuttuihin <rpc>- ja <rpc-reply>-sanomiin, joiden avulla ne saadaan yksilöityä NETCONF-sanomiksi. Nimiavaruuden lisäksi asiakas ja palvelin ilmoittavat tukemansa NETCONF-ominaisuudet URI-merkkijonoilla (Uniform Resource Identifier) kättelyn aikana. [11 s. 10.]

Konfiguraatio- ja tilatieto

NETCONF-laitteelta haettava tieto luokitellaan kahteen eri luokkaan: konfiguraatio- ja tilatietoon. *Konfiguraatiotieto* on kirjoitettavaa dataa, joka vaaditaan kun, halutaan muuttaa laitteen tila sen oletustilasta nykyiseen tilaan. *Tilatieto* on dataa, joka ei sisällä konfiguraatiotietoja, esimerkiksi järjestelmän verkkokortista tallentama data lähetetyistä ja vastaanotetuista paketeista, kuten kuviossa 10 on esitetty. NETCONF-operaatiokerroksen <get>-toimenpide tukee tilatietoja ja konfiguraatiotietoja. Muut operaatiokerroksen toimenpiteet tukevat ainoastaan konfiguraatiotietoja.

Konfiguraatioiden tietovarastot

NETCONF määrittelee laitteelle yhden tai useamman tietovaraston, johon laitteeseen määriteltävät asetukset, siis konfiguraatiotiedot, voidaan tallentaa. Tietovarastot voivat sijaita laitteen kovalevyllä, FLASH-muistissa tai vaikka FTP-palvelimella (File Transfer Protocol). NETCONF tukee kolmea erityyppistä tietovarastoa: aktiivista (running), käynnistyksessä käytettävää (startup) sekä kandidaatti-tietovarastoa (candidate). NETCONF-protokollan määrittelyssä todetaan, että palvelimen on tuettava ainakin aktiivista tietovarastoa. NETCONF-palvelin ilmoittaa tukemansa tietovarastot asiakkaalle ominaisuuksien vaihdon yhteydessä. [12, s. 7, 19-20; 9, s. 118.]

Hakutulosten suodatus

NETCONF-protokolla mahdollistaa operaatiokerroksen <get>- ja <get-config>-toimenpiteiden palauttamien <rpc-reply>-vastausten tietojen suodattamisen niin kutsutulla XML subtree -suodattamisella. XML subtree -suodatusta käyttämällä voidaan hakea vain haluttu osa konfiguraatio- tai tilatiedoista, jolloin saadaan kutistettua vastausviestin kokoa. XML subtree -suodatuksen kriteereinä voidaan käyttää muun muassa XML-nimiavaruutta. [8, s. 129-130.]

3.2 NETCONF-ominaisuudet (Capabilities)

NETCONF-asiakas ja -palvelin synkronoivat istunnossa käytettävät ominaisuudet lähettämällä toisilleen rinnakkaisesti sanoman, jossa kunkin tuetut ominaisuudet kuvataan. Tämä tapahtuu heti kun yhteys on muodostettu *kuljetuskerroksessa*. Sanoma koostuu <hello>-elementtiin koteloiduista URI-merkkijonoista, joissa jokaista ominaisuutta kuvaa yksi merkkijono. Merkkijono voidaan esittää joko URL- tai URN-koodattuna. Lisäksi palvelin ilmoittaa samalla NETCONF-istunnon tunnisteen <session-id>-elementillä, jota tarvitaan istunnon lopettamisen yhteydessä.

NETCONF-protkollan ydinominaisuudet koostuvat URN-merkkijonoista (Uniform Resource Name), kuten esimerkiksi seuraavassa merkkijonossa

```
urn:ietf:params:netconf:capability:{name}:1.0
```

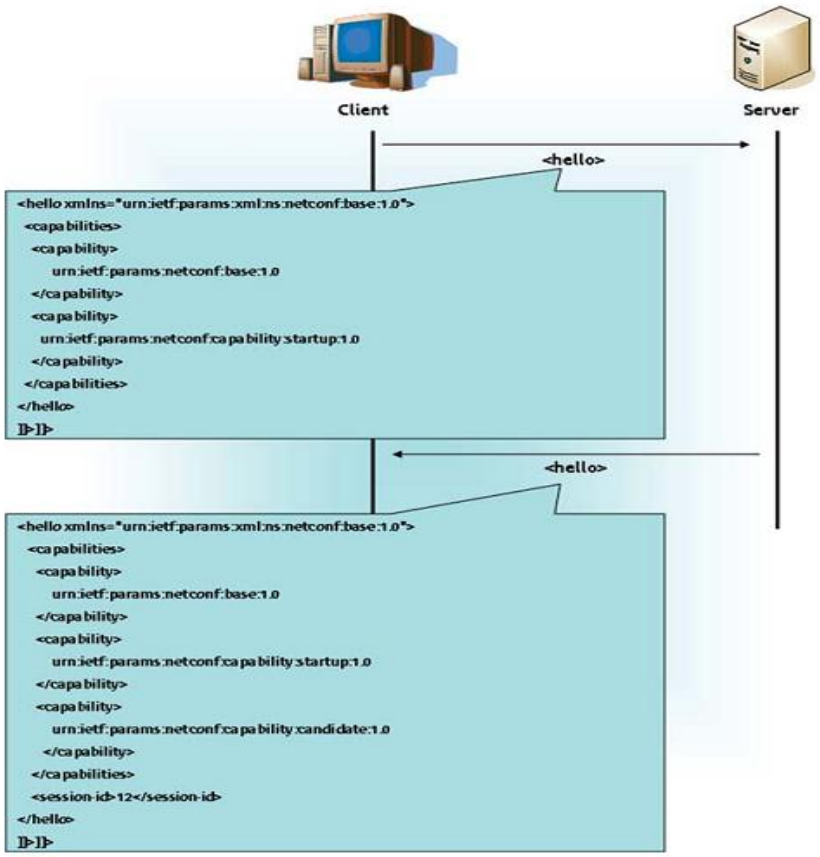
jossa {name} korvataan ominaisuuden nimellä. Ominaisuuksiin voidaan myös viitata käyttämällä lyhennettyä merkintää *:{name}*.

Esimerkiksi palvelimen kykyä validoida konfiguraation syntaksi voidaan ilmaista *:validate*-nimellä, joka on siis lyhennetty URN-merkinnästä

urn:ietf:params:netconf:capability:validate:1.0.

Laitteiden valmistajat voivat myös määritellä omia laitekohtaisia ominaisuuksiaan, jotka yksilöidään kyseisen valmistajan URN- tai URL-merkkijonolla (Uniform Resource Locator).

Kuvio 10 esittää palvelimen ja asiakkaan ominaisuuksien synkronoinnin SSH-yhteyden (Secure Shell) yli. Siinä asiakas ilmoittaa tukevansa NETCONFin versiota 1.0 sekä *:startup*-ominaisuutta, jotka on molemmat esitetty URN-merkkijoinoina. Palvelin tukee asiakkaan tukemien ominaisuuksien lisäksi *:candidate*-ominaisuutta (kandidaattitietovarastoja), mutta koska asiakas ei tue tätä ominaisuutta, sitä ei voida käyttää istunnossa. [8, s. 157.]



Kuvio 10. NETCONF-asiakas ja -palvelin synkronoivat istunnossa käytetyt ominaisuudet [8, s. 157].

Palvelimen tukemat ominaisuudet vaikuttavat NETCONF-operaatioden toimintaan. Jos esimerkiksi palvelin tukee *:candidate*-ominaisuutta, voidaan <edit-config>-operaatio kohdentaa kandidaatti-tietovarastossa sijaitsevaan konfiguraatioon, jolloin halutut asetukset voidaan kokonaisuudessaan määrittää, ja vasta silloin, kun kaikki asetukset on määritelty, ottaa konfiguraatio käyttöön lähettämällä palvelimelle <commit>-käsky. *:confirmed-commit*-ominaisuus on riippuvainen *:candidate*-ominaisuudesta, kaikki muut NETCONF 1.0 -versiossa määritellyt ominaisuudet ovat itsenäisiä, eivätkä täten ole riippuvaisia muista ominaisuuksista.

Taulukossa 1 on esitetty NETCONF version 1.0:n tukemat ominaisuudet RFC 4741 -dokumentin pohjalta. *:writetable-running-*, *:candidate-*, *:startup-* sekä *:url-* ominaisuudet liittyvät konfiguraatiotiedostojen tietovarastojen määrittelemiseen. Ne voidaan antaa lähde- tai kohdetiedostoiksi esimerkiksi <edit-config>-, <copy-config>- tai <delete-config>-toimenpiteille. *:confirmed-commit-* ja *:rollback-on-error-*

ominaisuuksilla voidaan vaikuttaa operaatioiden kulkuun virheen tai poikkeustilanteen syntyessä. [11, s. 50-63.]

Taulukko 1. NETCONF-version 1.0 ominaisuudet.

Ominaisuus	Kuvaus
:writable-running	Palvelin tukee suoria muokkaus- ja tallennusoperaatioita aktiiviseen tietovarastoon.
:candidate	Palvelin tukee erillistä kandidaatti-tietovarastoa. Tämän avulla muutokset voidaan kohdentaa erilliseen (ei aktiiviseen) tietovarastoon ja ottaa ne käyttöön <commit>-operaatiolla kun kaikki muutokset on tehty tai vaihtoehtoisesti hylätä muutokset <discard-changes>-operaatiolla.
:confirmed-commit	Palvelin tukee <confirmed>- ja <confirm-timeout>-parametrejä <commit>-operaatiolle. Jos :confirmed-commit-ominaisuutta käytetään, tulee asiakkaan lähettää <commit>-operaatio palvelimelle <confirm-timeout>-parametrissa määritellyn ajan puitteissa (oletusarvo 600 sekuntia). Jos <commit>-operaatiota ei lähetetä määräajassa, palvelimen tulee hylätä tehdyt muutokset. Muutoin muutokset siirretään aktiivisessa tietovarastossa sijaitsevaan konfiguraatioon.
:rollback-on-error	Palvelin tukee rollback-on-error -parametria <edit-config>-operaation <error-option>-parametrissa.
:validate	Palvelin tukee ominaisuutta tarkastaa kandidaatti-tietovarastoon tallennetusta konfiguraatiosta syntaktisia tai semanttisia virheitä.
:startup	Palvelin tukee erillistä startup-tietovarastoa, josta laite lataa konfiguraation käynnistyksen yhteydessä.
:url	Palvelin tukee operaatioita konfiguraatiotiedostoihin url-ominaisuudessa määritellyistä poluista. Mahdollisia arvoja ovat esimerkiksi HTTP, FILE ja FTP.
:xpath	Palvelin tukee hakutulosten rajaamista XPath-lausekkeilla.

3.3 NETCONF-protokollan kerrokset

Kuviossa 11 on esitettynä NETCONF-protokollan neljä kerrosta: *1. kuljetus-, 2. viesti-, 3. operaatio-* sekä *4. sisältökerros*. Kukin kerros huolehtii tietystä tarkkaan määritellystä osasta protokollan toiminnassa.

Kuljetuskerrosta ei ole sidottu yksittäiseen protokollaan. RFC 6264 määrittelee ainoastaan, että yhteyden on täytettävä seuraavat kriteerit:

- Toteuttavan mekanismin tulee todentaa yhteyden osapuolet (authentication).
- Mekanismin on varmistettava yhteyden eheys (integrity).
- Ratkaisun tulee taata yhteyden luottamuksellisuus (confidentiality).

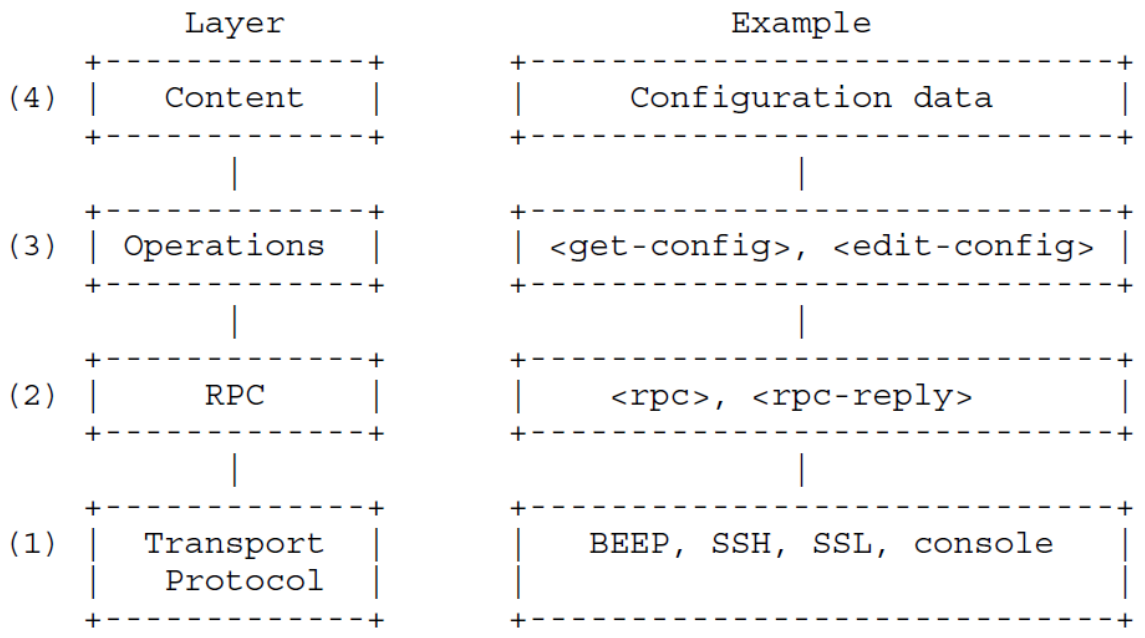
- Yhteyden on pysyttävä päällä koko NETCONF-istunnon keston ajan. Sen on siis oltava yhteydellinen (connection oriented).

NETCONF-protokollan toteuttavassa laitteessa voi olla tuki usealle protokollalle. NETCONF-standardi määrittelee, että ainoastaan tuki SSH-yhteyksille tulee toteuttaa. Tuki muita protokollia hyödyntäville yhteyksille on vapaaehtoinen.

RPC-kerros hyödyntää RPC-arkkitehtuuria viestien välityksessä hallintaohjelman (*NETCONF-asiakas*) sekä hallittavan laitteen (*NETCONF-palvelin*) välillä. Istunnon väliset osapuolet (asiakas ja palvelin) keskustelevat lähettämällä XML-koodattuja viestejä keskenään.

Operaatiokerros määrittele pienen joukon perusoperaatioita, niin kutsuttuja RPC-metodeja, joiden rakenne on määritelty XML-skeemassa. Yksi tällainen operaatio on esimerkiksi *<get>*-operaatio, joka lähettää tulostuspyynnön laitteen konfiguraatiolle tai sen osalle.

NETCONF-protokollan määrityksessä ei oteta kantaa *sisältökerroksessa* esitetyn tiedon formaattiin. Siinä välitetään palvelimelle lähetettävät toteutuskohtaiset komennot. IETF on perustanut NETMOD-työryhmän, jonka toimesta on julkaistu YANG-tietomalli NETCONF-protokollan tila- ja asetustietojen, RPC-kutsujen sekä ilmoitusten standardinmukaiselle esittämiselle. Sisältökerroksen YANG-tietomallia voitaisiin verrata vaikka SNMP-protokollan käyttämään SMI (Structure of Management Information) -malliin. [11, s. 6-7; 8, s. 106, 114-116.]



Kuvio 11. NETCONF-protokollan kerrokset [11, s. 6].

3.3.1 Kuljetuskerros

NETCONF-protokollan kuljetuskerrosta ei tule sekoittaa OSI-mallin (Open System Interconnection) kuljetuskerrokseen. Kyseessä on oikeastaan sovelluskerroksen alikerros. Kuvio 12 havainnollistaa sovelluskerroksen ja kuljetuskerroksen välistä yhteyttä. Kuljetuskerroksen vaatimuksena on *turvallinen* sekä *katkeamaton* yhteys. Istunnot saattavat olla pitkäkestoisia ja yhteyden tulee säilyä koko istunnon keston ajan. Yhteyden toteuttavan protokollan tulee myös tukea mahdollisuutta ilmaista istunnon tyyppi NETCONF-protokollalle, eli onko kyseessä asiakkaan vai palvelimen istunto.

4. Sisältökerros	YANG
3, Operaatiokerros	<get-config>, <lock>, <close-session>
2. Viestikerros	<rpc>, <rpc-reply>, <rpc-error>, <ok>
1. Kuljetuskerros	SSH, HTTPS, TLS, SOAP
7. Sovelluskerros	SSH, HTTPS, SMTP, FTP
6. Esityskerros	
5. Istuntokerros	
4. Kuljetuskerros	
3. Verkkokerros	
2. Siirtokerros	
1. Fyysinen kerros	

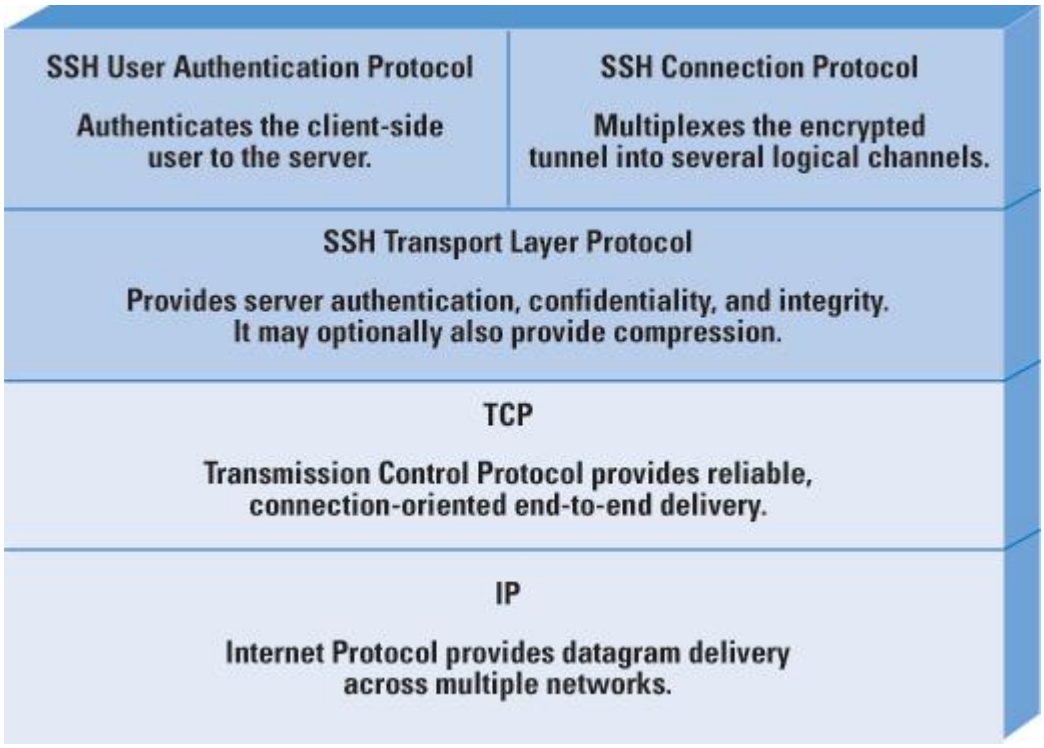
Kuvio 12. Kuljetuskerros on OSI-mallin sovelluskerroksen alikerros. [12, s. 9, 16, 19, 35; 13)

Jos NETCONF-istunto katkeaa riippumatta siitä, oliko syynä asiakkaan lähettämä keskeyttämiskäsky vai virhe yhteydessä, tulee palvelimen vapauttaa käytössä olleet resurssit. Resurssi voisi vaikkapa olla asetustiedostolle laitettu tiedostolukko, joka estää muita käyttäjiä tekemästä konfiguraatioon muutoksia niin kauan kuin tiedostolukko on aktiivisena tai NETCONF-palvelimelle kirjautunut käyttäjä.

Turvallisella yhteydellä viitataan yhteyteen, joka on vahvistettu, eheä, luottamuksellinen sekä tekee mahdottomaksi tiedon toistamisen. Tämä onnistuu käytännössä muodostamalla yhteys esimerkiksi SSH- tai TLS-protokollalla (Transport Layer Security). Yksi tärkeä tavoite NETCONF-protokollaa suunniteltaessa on ollut kehittää ohjelmoitava ratkaisu, joka seuraa läheisesti kohdejärjestelmän alkuperäistä liittymää. Tästä syystä on odotettavissa, että kohdejärjestelmä (NETCONF-palvelin) hyödyntää olemassa olevia menetelmiä autentikointiin. Esimerkiksi RADIUS (Remote Authentication Dial In User Service) -protokollaa todentamiseen tukeva päätelaite voisi käyttää samaa todentamista NETCONF-palvelimella. Todentamisen on johdettava tilaan, jossa palvelimella on selvyys kirjautuneesta käyttäjästä sekä tämän oikeuksista. Käyttäjien yksilöimiseen voidaan käyttää järjestelmän olemassa olevia käyttäjätunnuksia edellyttäen, että käyttäjätunnus on siinä muodossa, että se voidaan esittää XML-koodina. [12, s. 11-12.]

SSH-istuntoon pohjautuva NETCONF-yhteys

SSH on protokolla, jonka avulla voidaan luoda turvallinen yhteydellinen yhteys kahden osapuolen välillä, niin yksityisen kuin julkisenkin verkon yli. SSH-protokollan alkuperäinen versio, SSH1, kehitettiin korvaamaan palveluja kuten TELNET ja RLOGIN turvallisena vaihtoehtona komentojen etäsuorittamiselle. SSH-protokollan toisessa versiossa, SSH2:ssa, paranneltiin SSH version 1:n määriteltyjä ominaisuuksia. SSH:n dokumentaatio kattaa RFC 4250:sta 4256:en. SSH koostuu kolmesta protokollasta, jotka yleensä käyttävät Transmission Control Protocol (TCP) -protokollaa, kuten kuviossa 13 esitetään.

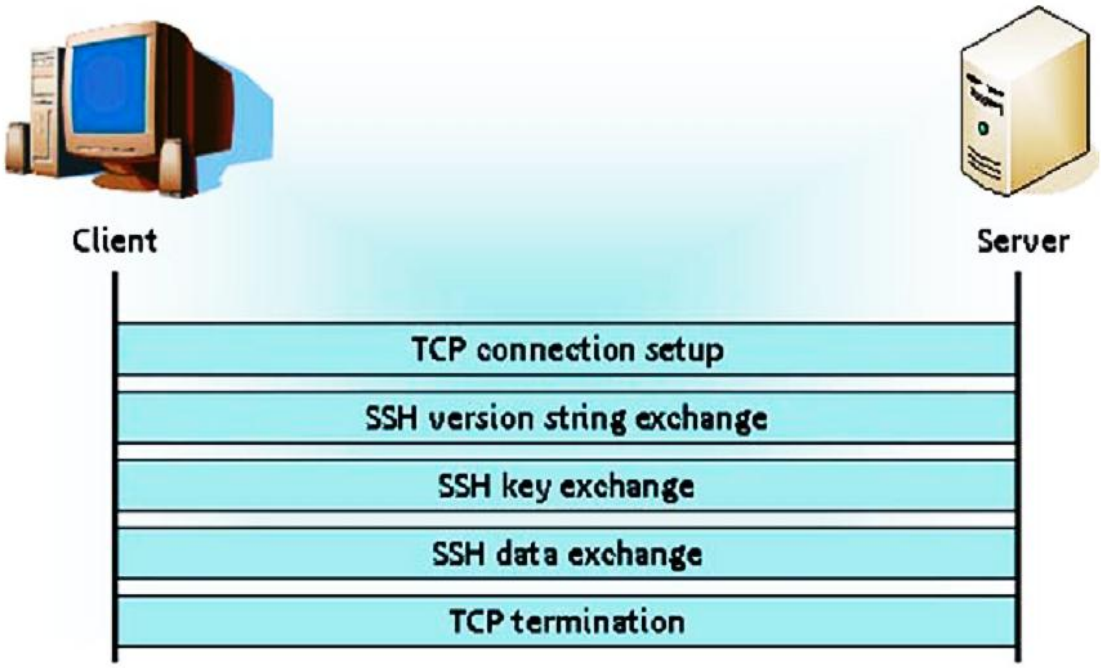


Kuvio 13. SSH2-protokollan kerrokset [14].

Kuviossa 13 esitetty *SSH-kuljetuskerroksen protokolla* (Transport Layer Protocol) käyttää yleensä TCP-protokollaa yhteyden muodostamiseen. Tämä kerros toteuttaa todennuksen (authentication), luottamuksellisuuden (confidentiality) sekä eheyden (integrity) varmistamisen. Kuvio 14 näyttää kuljetuskerroksen muodostuksen, joka koostuu seuraavista askelista:

- *TCP-yhteyden muodostus.* Yhteys muodostetaan SSH-palvelimen porttiin, jossa NETCONF-palvelin kuuntelee. Internet Assigned Numbers Authority (IANA) on määritellyt NETCONF-protokollalle TCP-portin 830.
- *SSH-versioiden synkronointi*. Molemmat osapuolet lähettävät tukemansa SSH-versiot. NETCONF vaatii SSH version 2:n toimiakseen.
- *SSH-avainten vaihto*. Osapuolet sopivat käytettävästä salausalgoritmista.
- *SSH-tiedon vaihto*. SSH-yhteys on avoinna ja osapuolet voivat lähettää dataa toisilleen.

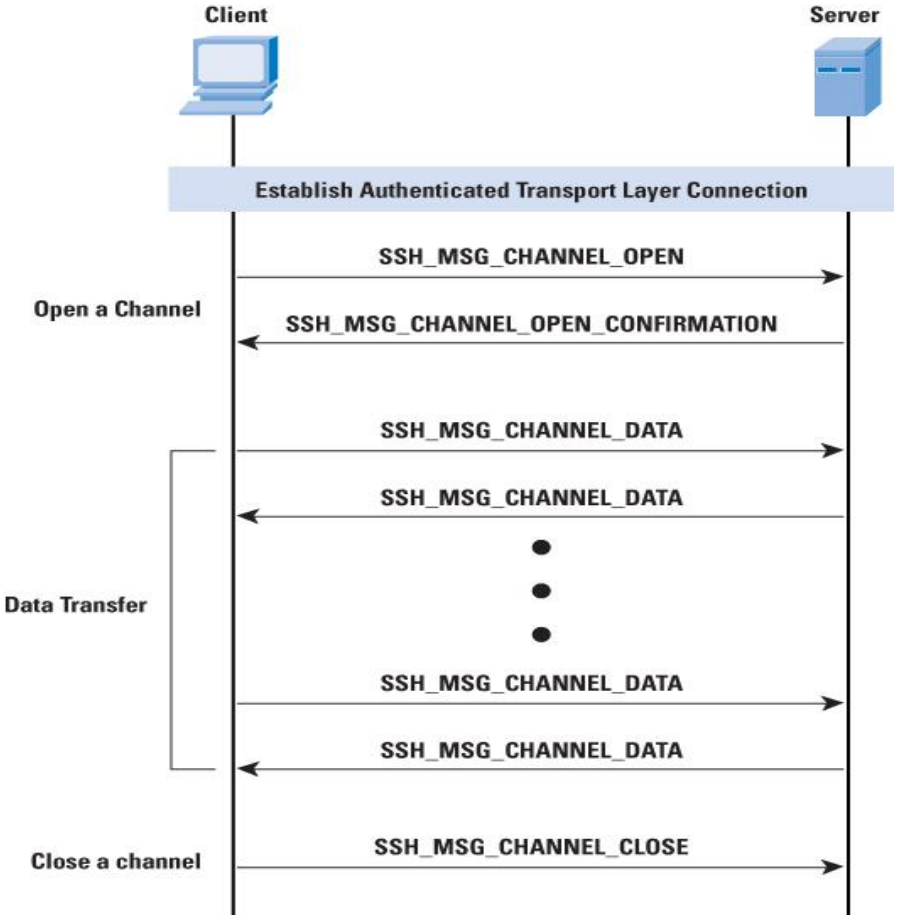
Istunnon päätyttyä yhteys katkaistaan. [8, s. 154.]



Kuvio 14. SSH-kuljetuskerroksen yhteyden muodostus [8, s. 155].

Kuviossa 13 *SSH-käyttäjätodennuksen protokolla* (User authentication protocol) vastaa asiakkaan todentamisesta palvelimelle. Siinä lähetetään viestit käyttäen SSH-kuljetuskerrosta. Käyttäjätodennus-protokollassa asiakas lähettää palvelimelle pyynnön, joka sisältää käyttäjätunnuksen, palvelun nimen sekä käytettävän todennusmetodin, joka on joko asiakkaan julkinen avain tai salasana. [14.]

SSH-yhteysprotokolla (SSH Connection Protocol) toimii SSH-yhteyden päällä, niin kutsutussa *SSH-tunnelissa*, joka on kuviossa 14 esitettynä SSH-tiedonvaihtona (SSH data exchange). Yhteysprotokolla muodostaa limittäin useita loogisia kanavia SSH-tunneliin. Kuviossa 15 on esitetty SSH-yhteysprotokollan toiminta. Sekä palvelin että asiakas voivat avata uuden kanavan. Palvelin ja asiakas tunnistavat käytettävän kanavan numeroimalla ne. Samalla kanavalla voi olla eri numerot palvelimella ja asiakkaalla [14].



Kuvio 15. Kanavien muodostus SSH-yhteysprotokollassa [15].

NETCONF-asiakas, joka käyttää SSH-protokollaa viestinvälityskerroksena, muodostaa yhteyden palvelimelle seuraavasti:

1. SSH-asiakas (jonka NETCONF-asiakas kutsuu) muodostaa SSH-kuljetuskerroksen yhteyden SSH-palvelimelle kuvan 15 mukaan.
2. SSH-istunnon osapuolet, asiakas ja palvelin, sopivat istunnossa käytetyistä salausavaimista.
3. SSH-palvelin käynnistää *ssh userauth* -palvelun käyttäjän todentamiseksi.

4. SSH-asiakas avaa yhteyden SSH-yhteysprotokollalla, jolloin SSH-palvelin käynnistää *ssh-connection*-palvelun.
5. Kun *ssh-connection*-palvelu on käynnistynyt, SSH-asiakas (NETCONF-asiakas) avaa SSH-kanavan. Tätä vaihetta kuvataan kuvion 16 alussa. Operaation tuloksena NETCONF-asiakkaalla on muodostettuna SSH-istunto NETCONF-palvelimelle ja NETCONF-kuljetuskerros (kuvio 13) on siis muodostettu.
6. NETCONF-palvelin lähettää palvelimelle kättelyviestin, jolla se synkronoi tuetut ominaisuudet (kuvio 10). [8, s. 156.]

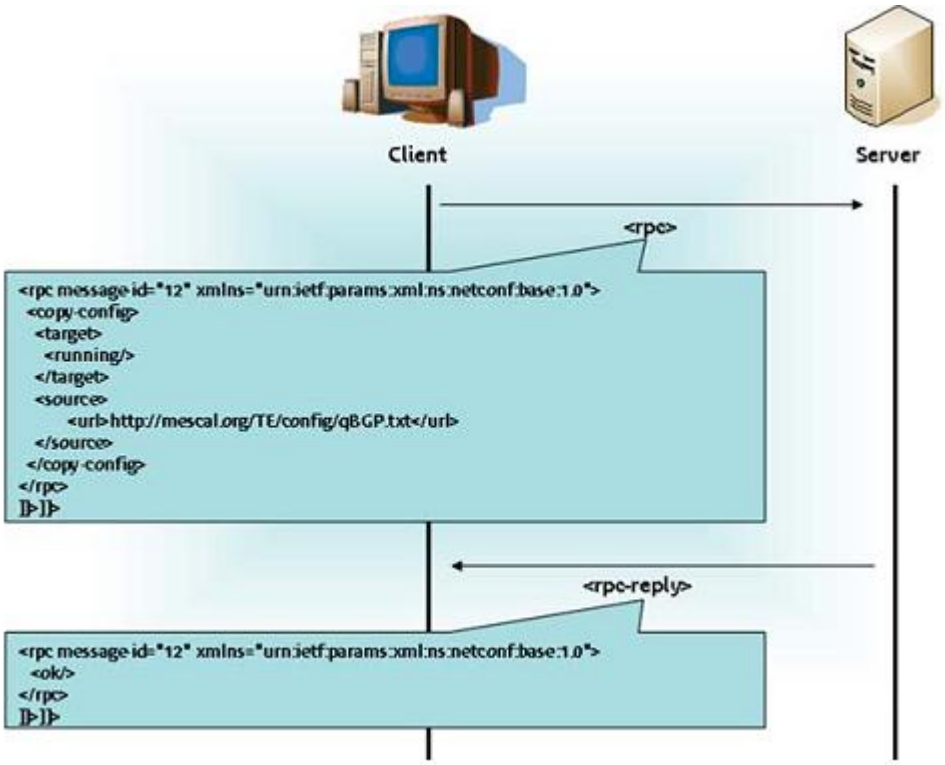
3.3.2 RPC-kerros

NETCONF-protokolla mahdollistaa palvelimen ja asiakkaan välisen viestien vaihdon kuljetusprotokollasta riippumattomalla tavalla toteuttamalla viestien koteloinnin mukautettuihin kehyksiin. Tämä niin kutsuttu Remote Procedure Call (RPC) -arkkitehtuuri kehystää asiakkaan lähettämät pyynnöt <rpc>-elementteihin ja palvelimen lähettämät vastaukset <rpc-reply>-elementteihin. Asiakkaan lähettämän <rpc>-elementein kehystetyn pyynnön ja palvelimen pyyntöön lähettämän <rpc-reply>-elementein kehystetyn vastauksen sitoo toisiinsa elementteihin määritelty attribuutti *message-id.* Asiakas määrittelee *message-id*-attribuutissa käytetyn arvon, jonka tulee olla jokin arvoa 4095 pienempi kokonaisluku. Palvelimen tulee vastauksessaan käyttää asiakkaan määrittelemää *message-id*-arvoa sellaisenaan.

Palvelimen <rpc-reply>-elementteihin kehystetty vastaus voi sisältää kolme arvoa. Ensinnäkin, jos asiakkaan lähettämä komento palauttaa palvelimelta dataa, sisällytetään data palvelimen vastaukseen. Toiseksi, jos lähetetty komento ei palauta dataa, mutta se suoritettiin onnistuneesti, palauttaa palvelin <rpc-reply>-elementteihin kehystetyn <ok>-elementin merkkinä onnistuneesta operaatiosta, kuten kuvion 7 kuvaamassa tilanteessa. Kolmanneksi, jos operaatio tuottaa virheen, palvelin palauttaa <rpc-reply>-elementteihin kehystetyn <rpc-error>-elementin, joka sisältää virheen tiedot elementin attribuuteissa.

Kuviossa 7 on esitetty <copy-config>-operaatio, jossa asiakas pyytää palvelinta kopioimaan WWW-palvelimella sijaitseva tiedoston HTTP-protokollan yli aktiivisessa tietovarastossa sijaitsevaan konfiguraatioon. <copy-config> on operaatiokerroksen komento, jonka onnistuessa palvelin palautta <ok>-elementin, kuten kuviossa 16, ja epäonnistu-

essa <error-reply>-elementin, jonka attribuuteissa on määritelty virheen tiedot. [8, s. 122-125.]



Kuvio 16. <copy-config>-operaation vaiheet. [8, s. 121].

3.3.3 Operaatiokerros

NETCONF-protokolla määrittelee rajatun määrän perustason toimenpiteitä konfiguraatio- ja tilatietojen hallintaan. Niiden avulla asiakas voi kutsua palvelimella suoritettavia operaatioita konfiguraatioiden tietovarastoille, kuten konfiguraatioiden tai sen osien noutoon, muokkaukseen, kopiointiin ja poistoon. Täydentäviä toimenpiteitä voidaan tukea julkistamalla tuetut toimenpiteet ominaisuuksina laitteiden kättelyn yhteydessä <hello>-pakettiin kehystettynä. [11, s. 31.]

Esimerkiksi Juniper Networksin NETCONF-toteutuksessa on suuri joukko ainoastaan Junos-käyttöjärjestelmää tukevia toimintoja. Junos-järjestelmän NETCONF-palvelin ilmoittaa täydentävistä ominaisuuksista <capability>-elementteihin kehystetyissä URL-merkkijonoissa, kuten seuraavasta esimerkistä ilmenee:

```
<hello>
<capabilities>
<capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
…
<capabil-
ity>http://xml.juniper.net/netconf/Junos/1.0</capability>
</capabilities>
<session-id>3911</session-id>
</hello>
```

[15, s. 40].

Esimerkistä on poistettu keskeltä rivejä kohdasta, jossa on kolme peräkkäistä pistettä (…), koska esimerkin tarkoituksena on esitellä valmistajakohtaisia ominaisuuksia. Esimerkki on muutoin täysin samankaltainen kuin esimerkiksi kuviossa 10 esitetty tilanne.

NETCONF versio 1.0 määrittelee yhdeksän perustason toimenpidettä, jotka on lueteltu taulukossa 2.

Taulukko 2. NETCONF-protokollan versiossa 1.0 määritellyt operaatiot.

Nimi	Kuvaus
<get-config>	Toiminto noutaa määrätyn konfiguraatiotiedoston tai sen osia.
<copy-config>	Toiminto kopioi konfiguraation tietovarastosta toiseen.
<edit-config>	Toiminto muokkaa määrättyä konfiguraatiotiedostoa tai sen osia.
<delete-config>	Toiminto poistaa kokonaisuudessaan määrätyn konfiguraation.
<lock>	Toiminto asettaa tiedostolukon määrätylle tietovarastolle.
<unlock>	Toiminto poistaa tiedostolukon määrätyltä tietovarastolta.
<get>	Toiminto palauttaa määrättyjä laitteen tilatietoja.
<close-session>	Toiminto katkaisee NETCONF-istunnon siten, että resurssit vapautetaan ja muut katkaisuun liittyvät toimenpiteet suoritetaan.
<kill-session>	Toiminto pakottaa NETCONF-istunnon katkaisun saman tien.

Taulukossa 2 luetellut operaatiokerroksen toimenpiteet käsittelevät lähes kaikki konfiguraatioiden tietovarastoja poikkeuksina <close-session>- sekä <kill-session>-toimenpiteet joita, käytetään NETCONF-istunnon sulkemiseen. Tietovarastoja muokkaaville toimenpiteille annetaan niiden käsittelemät tietovarastot parametreilla lähdetietovarasto (<source>) sekä kohdetietovarasto (<target>). Toimenpide saattaa vaatia jommankumman tietovarastoista tai molemmat.

<get-config>-toimenpiteellä noudetaan lähdetietovarastosta joko konfiguraatio kokonaisuudessaan tai konfiguraation osia riippuen siitä, annetaanko toimenpiteelle para-

metrina suodatuskäsky. Toimenpide palauttaa määrätyn haun tulokset onnistuessaan, ja epäonnistuessaan virheen, joka ilmaistaan <rpc-error>-elementissä. [11, s. 31-32.]

<copy-config>-toimenpide kopioi parametreina saamistaan tietovarastoista konfiguraatiot lähdetietovarastosta kohdetietovarastoon. Jos kohdetietovarastosta löytyy ennestään tietoa, korvautuu tieto lähteestä saadulla sisällöllä. Toimenpide palauttaa <ok>-elementin toimenpiteen onnistuessa, ja sen epäonnistuessa <rpc-error>-elementin. Esimerkki <copy-config>-toimenpiteestä on nähtävissä kuviossa 16. [11, s. 39-40.]

<edit-config>-toimenpiteelle annetaan parametreiksi lähde- ja kohdetietovarastot samoin kuin <copy-config>-toimenpiteelle. <edit-config>-toimenpide eroaa <copy-config>-toimenpiteestä siinä, että sillä voidaan kopioida ainoastaan tietty osa konfiguraatiota lähteestä kohteeseen. <edit-config>-toimenpiteelle voidaan lisäksi määritellä useita vaihtoehtoja siitä, miten sen tulisi toimia tilanteessa, jossa kohdetietovarastossa sijaitsevasta konfiguraatiosta löytyy ennestään lähdettä vastaavat tiedot. <edit-config>-toimenpide tukee lisäksi ominaisuutta, jolla voidaan testata toimenpiteestä aiheutuvat seuraukset ennen kuin toimenpide suoritetaan. <edit-config> palauttaa <rpc-reply>-elementissä <ok>-elementin, jos toimenpide onnistuu, ja sen epäonnistuessa <rpc-error>-elementin. [11, s. 34-37.] <delete-config>-toimenpide poistaa parametrina saamansa konfiguraation tietovarastosta.

Virheellisessä tilanteessa palvelin palauttaa <rpc-error>-elementin ja onnistuessaan <ok>-elementin.

<lock>- sekä <unlock>-toimenpiteet asettavat parametreina saamansa tietovarastot lukituiksi niin, ettei kukaan muu voi muokata niitä niin kauan kuin, lukitus on voimassa. Tällä saadaan estetyksi muita muokkaamasta tietovarastoja samaan aikaan kun ne ovat omalla asiakkaalla käytössä. Lukitus vaikuttaa kaikkiin muihin käyttäjiin, riippumatta siitä, onko kyseessä toinen NETCONF-istunto vai esimerkiksi komentoriviä tai jotain muuta liittymää hyödyntävä osapuoli. Molemmat toimenpiteet, <lock> ja <unlock>, palauttavat <ok>-elementin toimenpiteen onnistuessa, ja virheen sattuessa <rpc-error>-elementin. Jos asiakkaan muodostama istunto palvelimelle jostain syystä katkeaa, tulee palvelimen poistaa kaikki asiakkaalla käytössä olleet tiedostolukot, huolimatta siitä, onko toimenpide valmis. Lukitusta ei voida myöntää tietovarastolle, jos

jokin seuraavista ehdoista on tosi:

- Lukitusta ollaan asettamassa kandidaatti-tietovarastolle, joka sisältää muutoksia, joita ei ole tallennettu.
- Jokin toinen sovellus tai istunto on asettanut tietovarastolle lukituksen.

Lukitus voidaan tarvittaessa pakottaa poistettavaksi esimerkiksi katkaisemalla lukitusta käyttävä istunto kutsumalla <kill-session>-toimenpide. Esimerkki asiakkaan palvelimelle lähettämästä aktiivisen konfiguraation (running) lukitsemispyynnöstä on

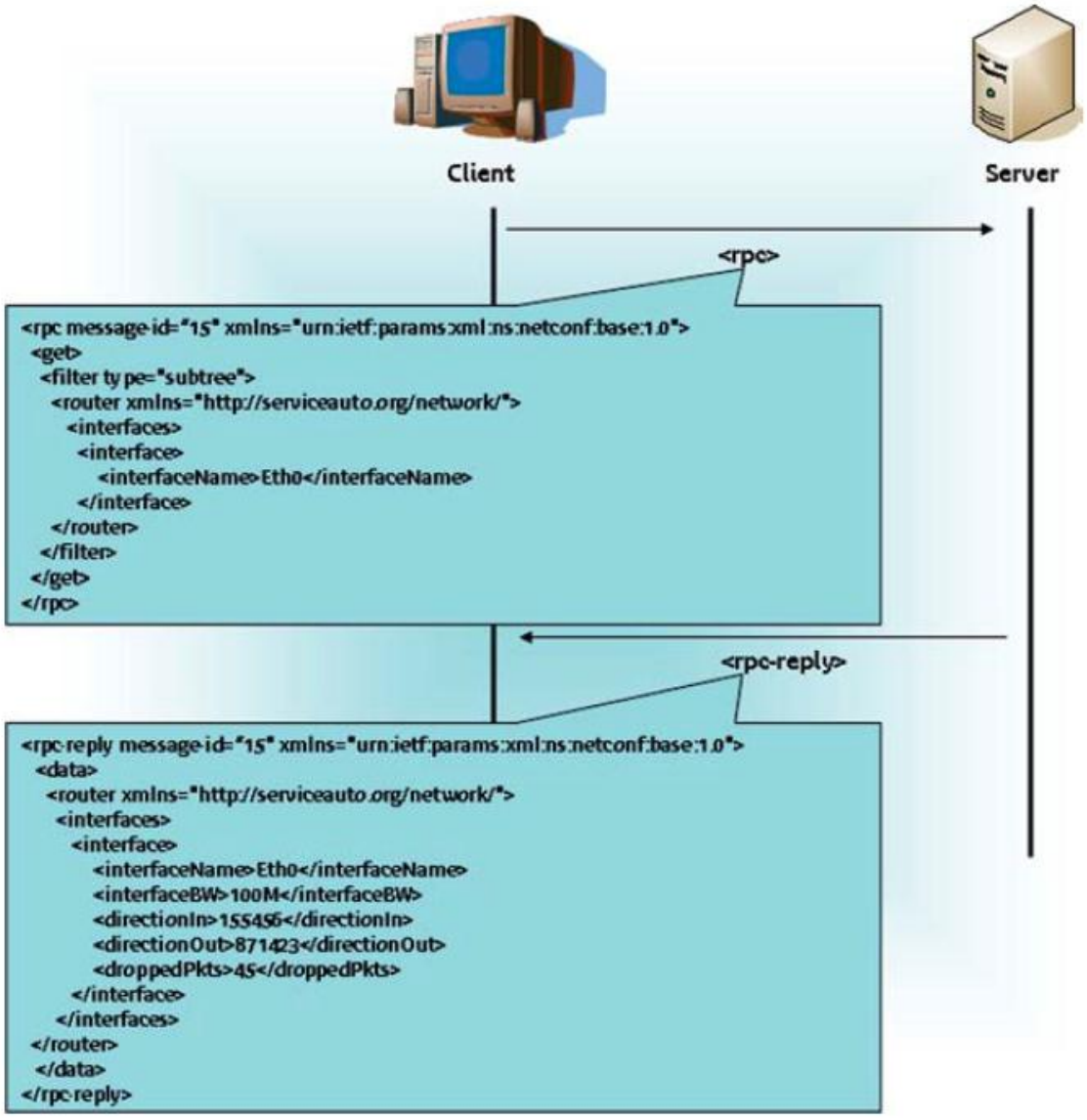
```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <lock>
    <target>
      <running/>
    </target>
  </lock>
</rpc>
```

Edellistä esimerkkiä jatkaen, olettaen, että aktiiviselle tietovarastolle olisi jo sijoitettu lukitus, seuraava esimerkki kuvaa palvelimen asiakkaalle lähettämää <rpc-error>-elementin sisältämää virheilmoitusta toimenpiteen epäonnistumisen seurauksena:

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>protocol</error-type>
    <error-tag>lock-denied</error-tag>
    <error-severity>error</error-severity>
    <error-message>
      Lock failed, lock is already held
    </error-message>
    <error-info>
      <session-id>454</session-id>
      <!-- lock is held by NETCONF session 454 -->
    </error-info>
  </rpc-error>
</rpc-reply>
```

<rpc-error>-elementin virheestä voidaan nähdä, että toisella NETCONF-istunnolla on jo asetettuna lukitus asiakkaan lähettämässä pyynnössä määriteltyyn tietovarastoon. [11, s. 42-44.]

<get>-toimenpidettä voidaan käyttää palauttamaan palvelimelta konfiguraation tietoja, samoin kuin <get-config>-toimenpidettä. <get>-toimenpide eroaa <get-config>-toimenpiteestä siten, että <get> mahdollistaa konfiguraatiotietojen lisäksi laitteen tilatietojen näyttämisen. Esimerkiksi kuviossa 17 on esitetty asiakkaan lähettämä <get>-pyyntö palvelimelle, josta suodatetaan kaikki paitsi palvelimen Eth0-verkkokortin tiedot.



Kuvio 17. <get>-toimenpide noutaa tiedot palvelimen Eth0-verkkokortista [8, s. 138].

Kuviossa 17 esitetyssä <rpc-reply>-viestissä on lueteltu palvelimen Eth0-verkkokortin aktiivinen nopeus sekä laskurit sen vastaanottamista ja lähettämistä paketeista.

<close-session>- ja <kill-session>-toimenpiteitä käytetään lopettamaan NETCONF-istunto. Ne saavat parametrina <session-id>-attribuutin, josta ilmenee, mikä istunto niiden tulisi sulkea. <kill-session>-toimenpidettä ei voida käyttää sulkemaan samaa istuntoa josta, se on kutsuttu. <close-session> ja <kill-session> palauttavat <ok>-attribuutin toimenpiteen onnistuessa ja <rpc-error>-attribuutin virheen sattuessa.

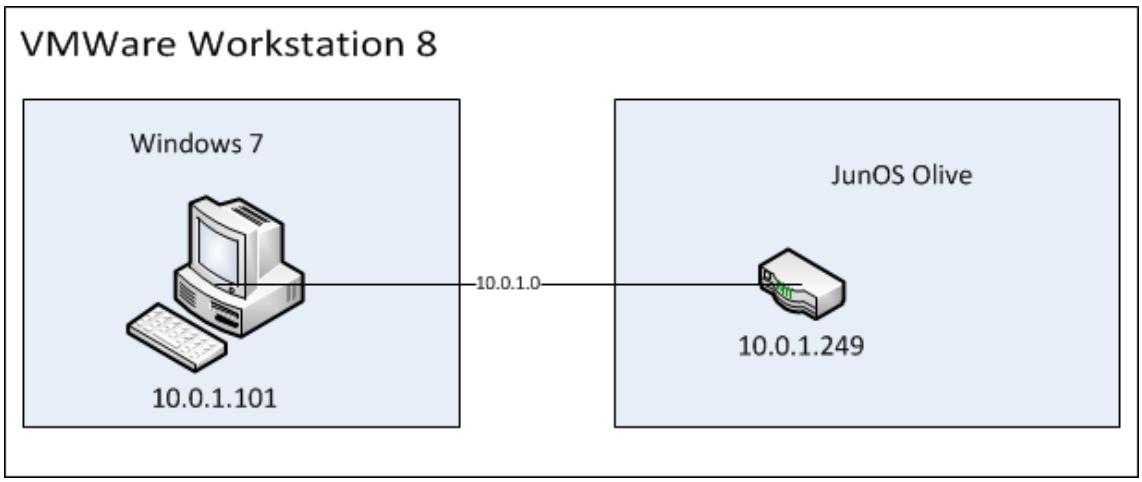
3.3.4 Sisältökerros

NETCONFin version 1.0 määrityksessä [11] ei oteta kantaa sisältökerroksen esitysmuotoon [8, s. 173]. IETF on muodostanut erillisen NETMOD-työryhmän, jonka tehtävänä on määrittää NETCONF-protokollalle standardin mukainen sisältökerroksen esitysmuoto. Työryhmä on julkaissut RFC 6020 -dokumentin, jossa määritellään YANG-niminen tiedonmallinnuskieli, jonka avulla voidaan määritellä konfiguraatiotiedolle standardin mukainen esitys. [16.]

4 Konfiguraation hallintasovellus

4.1 Testiympäristö

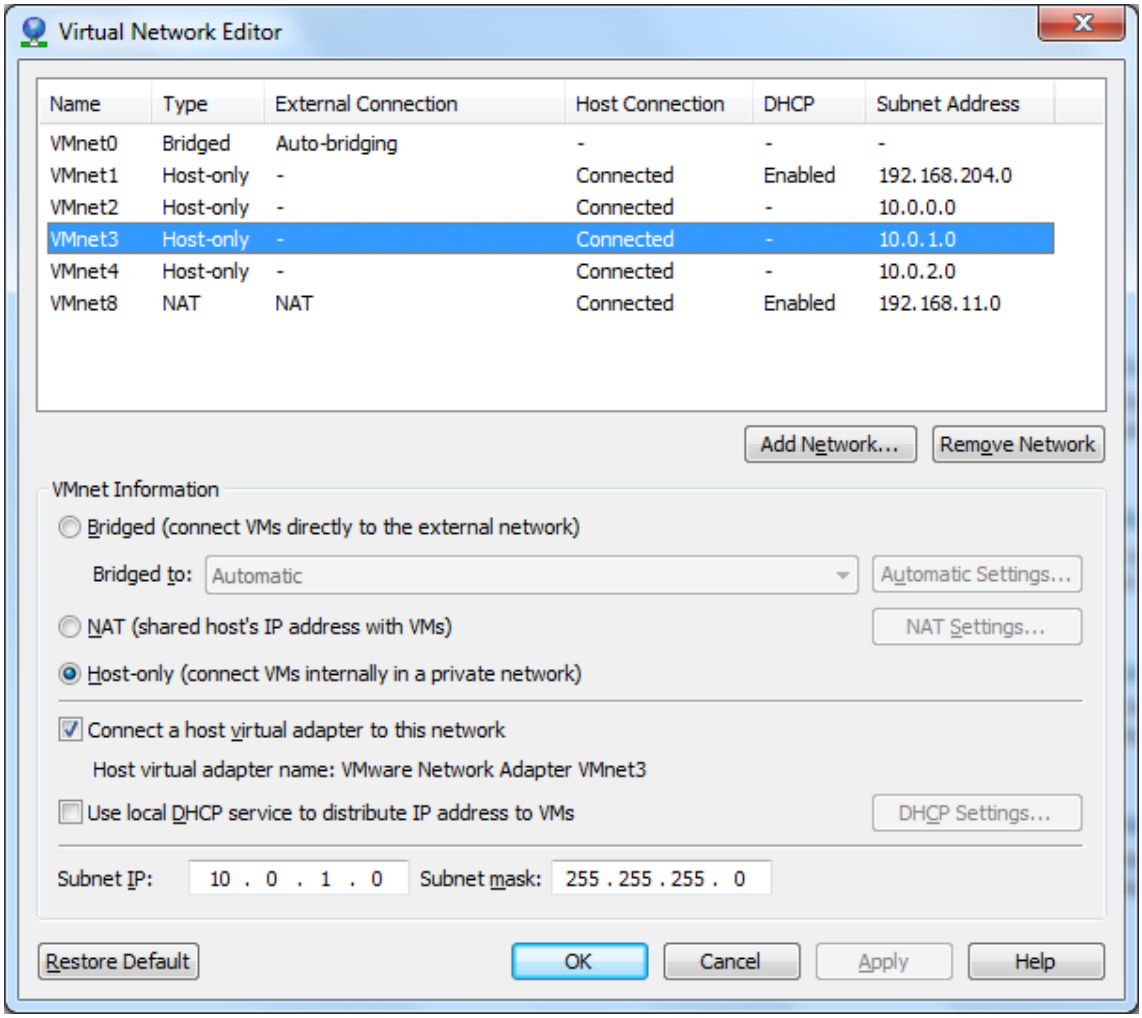
Opinnäytetyötä varten rakennettiin VMware Workstation -ohjelmalla toimiva testiympäristö. Kuvio 18 havainnollistaa testiympäristöä. Testiympäristö koostui Windows 7 -virtuaalikoneesta sekä Olive-emulaattorilla toimivasta Juniperin Junos-käyttöjärjestelmästä.



Kuvio 18. Opinnäytetyössä käytetty testiympäristö.

VMware Workstation on käyttöjärjestelmien virtuaalistamiseen kehitetty ohjelma. Sitä voidaan hyödyntää muun muassa testauksessa, ohjelmistojen kehityksessä, järjestelmien integrointiin liittyvissä hankkeissa sekä opetuksessa. VMware Workstation -sovelluksen ominaisuudet ovat suunnattuja työasemakäyttöön. [17, s. 6-7.]

Testiympäristön virtuaalikoneet yhdistettiin toisiinsa VMware Workstation -sovelluksessa luodulla virtuaaliverkolla. Kuviossa 19 on esitetty järjestelyssä käytetyt verkkoasetukset. Testiympäristölle nimettiin VMnet3-verkko, jossa kullekin laitteelle määriteltiin IP-asetukset manuaalisesti. Workstation-sovelluksen suljetun verkon kokoonpanossa verkon jäsenet näkevät ainoastaan toisensa [17, s. 11]. Tämä saavutettiin asettamalla verkkoeditorista päälle kytkentä *Host-only* kuvion 19 mukaan.



Kuvio 19. Testiympäristön verkkoasetukset. Näkymä VMware Workstation -ohjelman verkkoasetuksista.

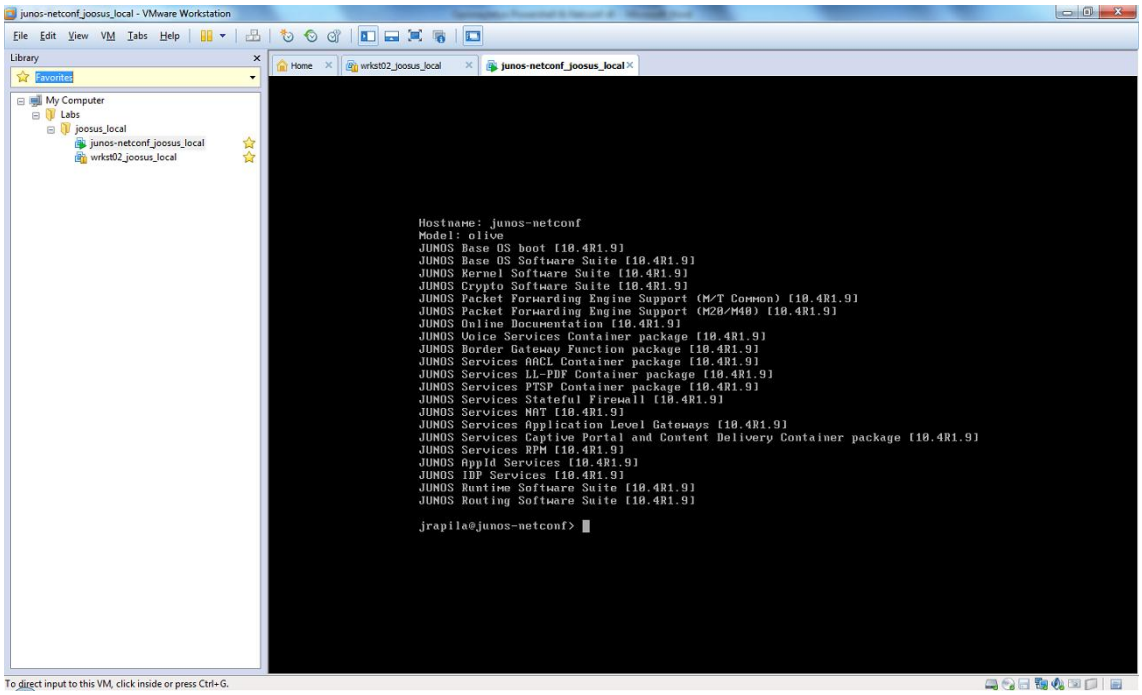
Testiympäristössä päädyttiin käyttämään Juniperin Junos-järjestelmää NETCONF-palvelimena, koska Junosin NETCONF-toteutuksesta löytyy hyvin dokumentaatiota. Junos on Juniperin 90-luvun alussa kehittämä, avoimen lähdekoodin FreeBSD-käyttöjärjestelmään pohjautuva, käyttöjärjestelmä. Junos on kehitetty alustaksi reitittimille, kytkimille ja muille verkkolaitteille. Junos eroaa muista yleisesti käytetyistä verkkolaitteiden käyttöjärjestelmistä siten, että se tarjoaa modulaarisen alustan, jossa järjestelmän ydin ja muut palvelut (kuten esimerkiksi eri reititysprotokollia suorittavat ohjelmat) ajetaan erillisinä prosesseina. Esimerkiksi Ciscon IOS-käyttöjärjestelmä on arkkitehtuuriltaan yksi, monoliittinen ohjelma. Ciscon IOS:lle toteuttama lähestymistapa mahdollistaa nopeamman suorittamisen, mutta altistaa samalla järjestelmän haavoittuvuuksille. Lisäksi, jos jokin järjestelmän komponentti vikaantuu, saattaa se kaataa koko järjestelmän. Junos-järjestelmässä tätä vaaraa ei ole olemassa. Junos-järjestelmässä on lisäksi panostettu siihen, että eri laitteille (reitittimet, kytkimet ja niin edelleen) käännetyt käyttöjärjestelmäversiot sisältävät samat ominaisuudet. Tällä on voitu taata yhtenäinen käyttökokemus käyttäjille laitteesta riippumatta. [18, s. 72-73.]

Olive

Olive on Juniperin alun perin yrityksen sisäiseen testikäyttöön kehittämä alusta, jonka avulla voidaan suorittaa Junos-käyttöjärjestelmää normaalilla PC-tietokoneella [19]. Juniper ei tarjoa tuotteelle virallista tukea, eikä sitä suositella ajettavaksi tuotantoympäristöissä.

Oliven asennus tapahtui siten, että ensiksi asennettiin FreeBSD-käyttöjärjestelmä VMware Workstation -sovellukselle. Asennuksessa valitaan asennettavaksi minimimäärä paketteja, jotka vaaditaan käyttöjärjestelmän käynnistämiseen. Opinnäytetyössä käytettiin *FreeBSD 4.11-RELEASE-i386-miniinst* -asennusta. FreeBSD:n asennuksen jälkeen asennettiin Junos-käyttöjärjestelmä. Asennus suoritettiin siirtämällä Windows 7 -virtuaalikoneella sijaitsevalta FTP-palvelimelta käyttöjärjestelmän asennustiedostot (jinstall-10.4R4.5-domestic-signed) FreeBSD-järjestelmään. Tämän jälkeen asennettiin Junos-järjestelmän asennustiedostot FreeBSD-järjestelmän pkg_add -komennolla. Kuviossa 20 on näkymä VMware Workstation -ohjelmassa käynnistetystä Junos Olive -

virtuaalikoneesta. Kuviosta 20 näkyy, että olive-emulointi on käytössä (kuvion toisella rivillä sijaitseva Model-teksti).



Kuvio 20. Junos-käyttöjärjestelmä käynnistettynä VMware Workstation -sovellukseen.

Käyttöjärjestelmän asennuksen jälkeen virtuaalikoneelle (Olive) määriteltiin verkkoasetukset, käyttäjät ja kytkettiin NETCONF-palvelin päälle. Liitteessä 1 on lopputyössä käytetty konfiguraatio Junos NETCONF -palvelimelle.

Yhdistettäessä komentoriviltä Junos-käyttöjärjestelmään on mahdollista suorittaa käsky, jonka avulla nähdään, mikä XML-koodattu komento vastaa jotakin normaalia komentoriviltä kutsuttavaa komentoa [18, s. 108]. Kuviossa 21 on esitetty komennon suorittaminen komentoriviltä sekä komennon tuottama tuloste normaalina sekä XML-koodattuna. Kuviossa suoritetaan ensin *show version* -komento ilman erillistä tulosteen muuntamista XML-muotoon. Jälkimmäisessä kohdassa suoritetaan samainen komento, mutta tällä kertaa komennon tuloste putkitetaan *display xml* -komennolle.

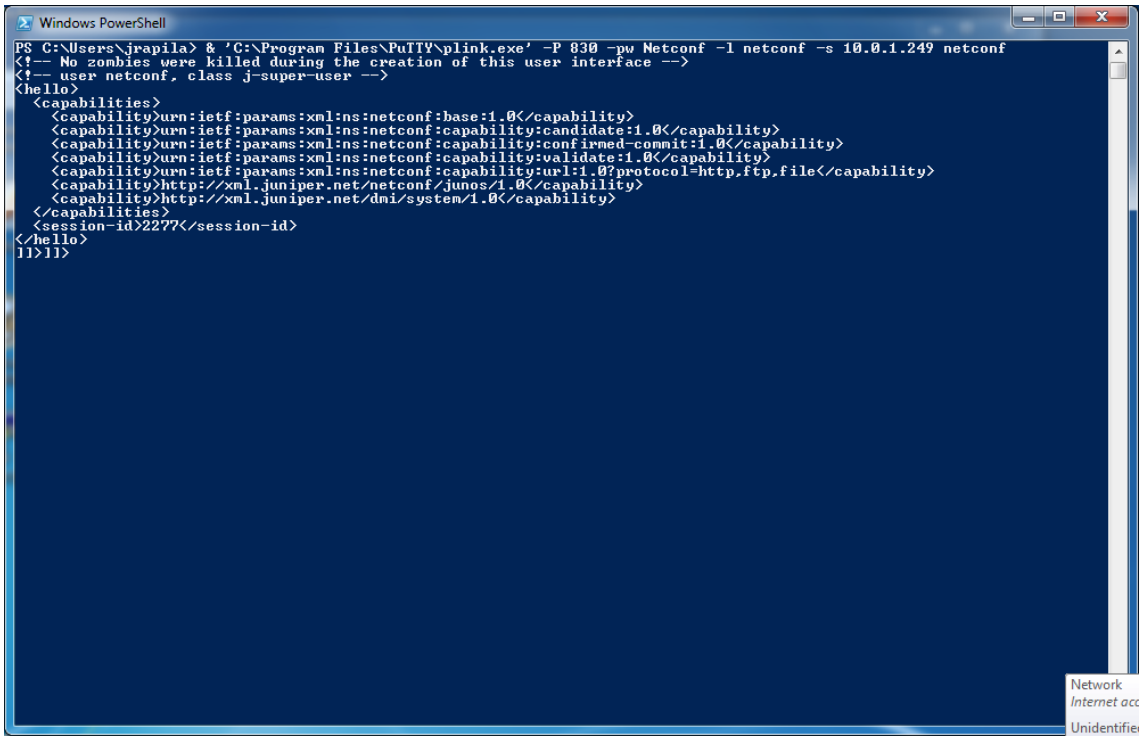
```
root@SRX210-A> show version
Hostname: SRX210-A
Model: srx210h
JUNOS Software Release [10.2B1.9]

root@SRX210-A> show version | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.2B1/junos">
    <software-information>
        <host-name>SRX210-A</host-name>
        <product-model>srx210h</product-model>
        <product-name>srx210h</product-name>
        <jsr/>
        <package-information>
            <name>junos</name>
            <comment>JUNOS Software Release [10.2B1.9]</comment>
        </package-information>
    </software-information>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>

root@SRX210-A>
```

Kuvio 21. Esimerkki komennon suorittamisesta Junos-järjestelmän komentoriviltä, XML-koodauksella ja ilman.

Kuvion 21 mukaista komentojen näyttämistä XML-koodattuina voidaan hyödyntää suoraan NETCONF-palvelimelle lähettävissä komennoissa. Kuviossa 22 on ruutukaappaus NETCONF-palvelimelle muodostetusta yhteydestä, joka on muodostettu putty-ohjelman mukana toimitettavalla plink.exe-ohjelmalla. Ohjelman kutsussa käytetyillä parametreilla määritellään käyttäjä (-l netconf), salasana (-pw Netconf), portti (-P 830), SSH subsystem -komento (-s netconf) ja Junos (Olive) -palvelimen osoite (10.0.1.249).

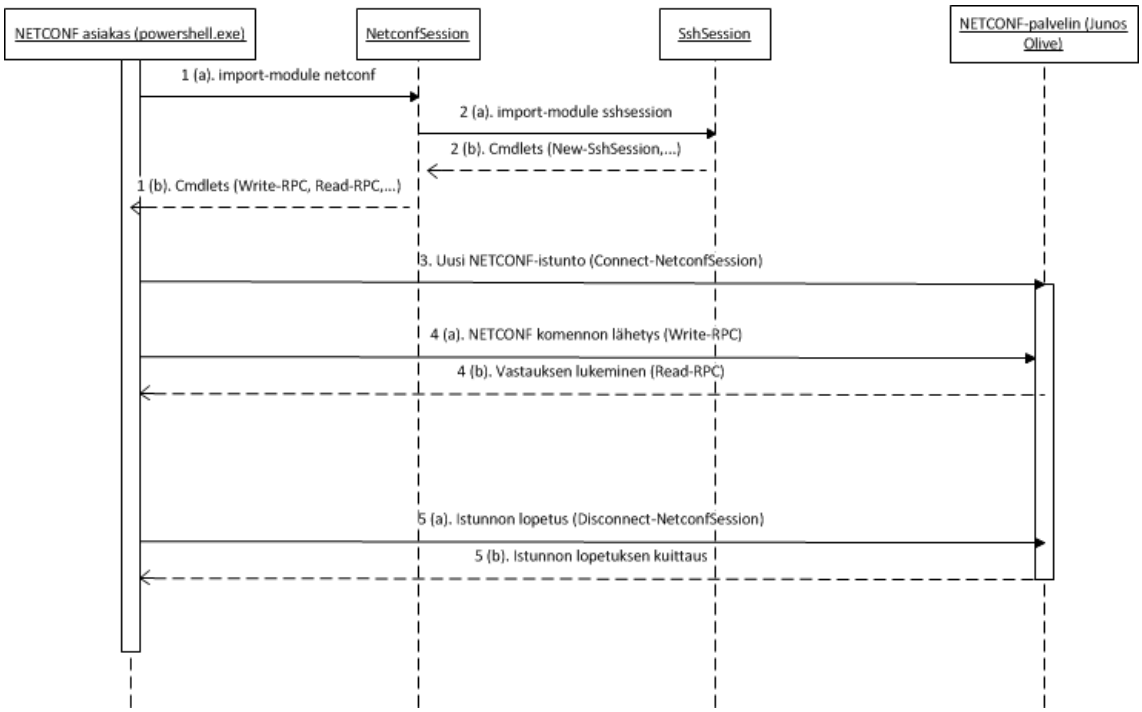


Kuvio 22. NETCONF-yhteyden muodostus plink.exe-ohjelmalla.

4.2 Powershell-asiakassovellus

NETCONF-asiakassovellus kehitettiin täysin käyttäen Powershell-skriptikieltä, lukuunottamatta NETCONF-kuljetuskerrosta (SSH), johon käytettiin Putty-ohjelman mukana toimitettavaa plink.exe-ohjelmaa. Editorina kehityksessä käytettiin Microsoftin Powershell ISE -isäntäsovellusta.

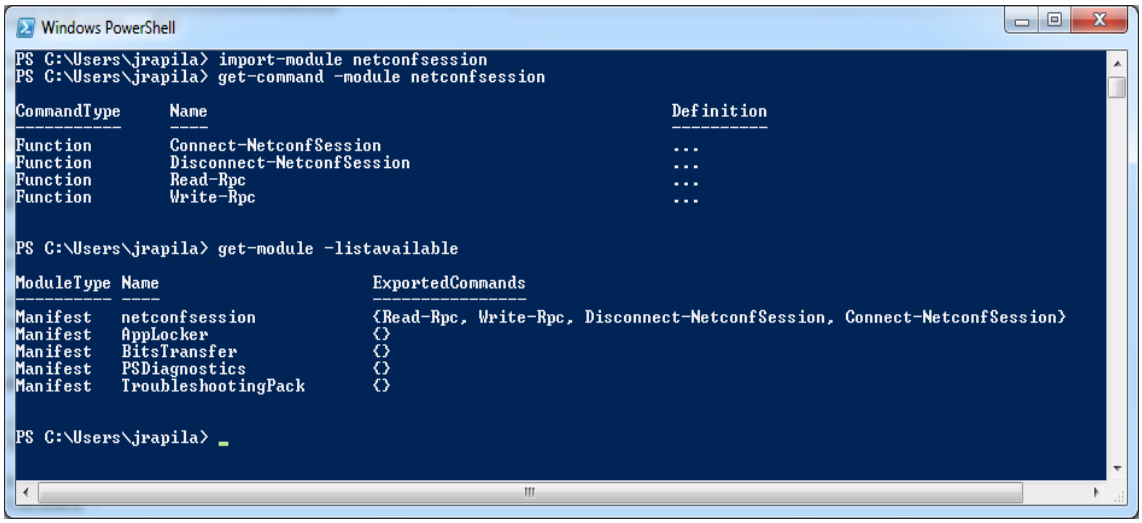
Kuviossa 23 on esitetty UML-mallintamiskielen sekvenssikaaviona ohjelman toiminta korkealla tasolla tarkasteltuna. NETCONF-asiakkaana käytettiin opinnäytetyössä powershell.exe-isäntäsovellusta, josta suoritettiin osana opinnäytetyötä tehdyt Powershell-skriptit. NETCONF-palvelimen rooli toteutettiin Junos Olive -sovelluksella. Asiakkaan ja palvelimen välisestä viestien välityksestä vastasivat sshsession- ja netconf-session-Powershell-moduulit.



Kuvio 23. NETCONF-asiakassovelluksen toiminta vaiheittain esitettynä.

Kuvion 23 ensimmäisessä vaiheessa, kohdat 1 (a) ja 1 (b), käyttäjä tuo netconfsession-moduulissa määritellyt komennot istuntoonsa komennolla *import-module netconfsession*. Käyttäjä näkee ainoastaan netconfsession-moduulin mutta ei sshsession-moduulia. Tämä johtuu siitä, että sshsession-moduuli on netconfsession-moduulin sisäkkäinen moduuli, joka sisältää ainoastaan tukifunktioita liittyen NETCONF-protokollan transport-kerroksen (SSH-yhteyden) hallintaan.

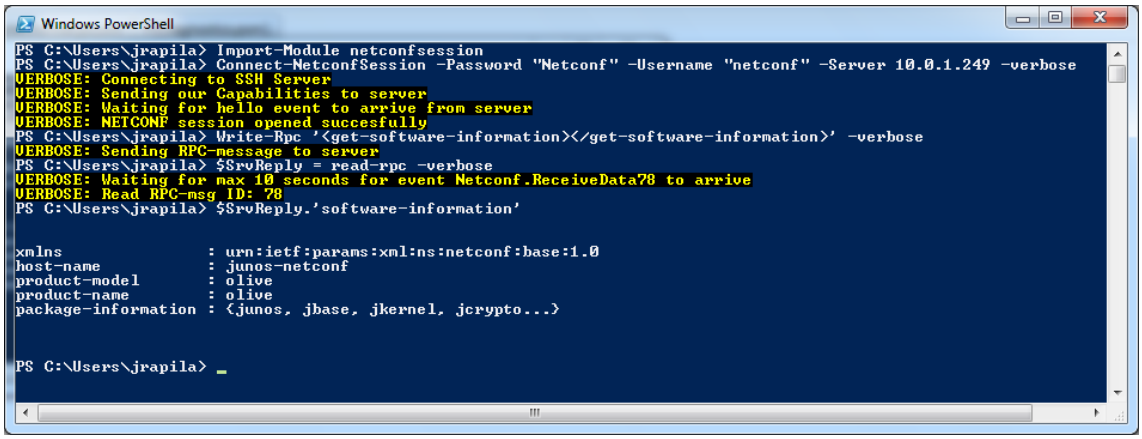
Kuviossa 24 on demonstroitu moduulin tuominen istuntoon. Siinä kuvattiin netconfsession-moduulin komentojen tuonti nykyiseen istuntoon (komennolla *import-module netconfsession*). Lisäksi tarkistettiin, että ainoastaan netconfsession-moduulissa määritellyt komennot olivat saatavilla komennolla *get-command -module netconfsession*, ja ettei käyttäjälle näkynyt sshsession-moduulissa määriteltyjä funktioita, jotka suoritettiin komennolla *get-module -listavailable*.



Kuvio 24. NETCONF-asiakkaan toteuttavan moduulin tuominen käyttäjän aktiiviseen Powershell-istuntoon.

Kuviossa 25 demonstroitiin kuviossa 23 esiteltyjen toimintojen suoritus. Aluksi tuotiin moduulit istuntoon komennolla *import-module netconfsession*. Komentojen suoritus vastasi kuvion 23 kohtia 1 (a) ja 1 (b). Sshsession-moduuli kutsuttiin netconfsession-moduulista, joita vastasivat kuvion 23 kohdat 2 (a) ja 2 (b).

Kuvion 23 kohdan 3 NETCONF-istunnon muodostus demonstroitiin kuviossa 25 toisella rivillä kutsutulla komennolla *connect-netconfsession*. *Connect-netconfsession*-komennolle syötettiin parametreinä NETCONF-palvelimen tiedot. *Connect-netconfsession*-komennosta kutsuttiin (sshsession-moduulista) *new-sshsession*-funktiota, jossa käynnistettiin plink.exe-ohjelma .NET-sovelluskehyksen System.Diagnostics.Process-luokan ilmentymänä.



Kuvio 25. Yhteyden muodostus NETCONF-palvelimelle ja kuviossa 21 esitetyn komennon suorittaminen NETCONF-palvelimella.

New-sshsession-funktiossa määriteltiin lisäksi asynkroninen funktio palvelimen syötteiden lukemiseen ja niihin reagointiin. Toiminto toteutettiin siten, että kun palvelin kirjoitti jotain asiakasohjelmalle, se näkyi System.Diagnostics.Process-luokaan pohjautuvalle oliolle saapuvana syötteenä. Sisään tulevaa syötettä valvottiin *Register-ObjectEvent*-ohjelmalla, joka on Powershell-ympäristön mukana toimitettava cmdlet-ohjelma, jonka avulla voidaan määritellä .NET-kehykseen pohjautuva olio reagoimaan tiettyyn tapahtumaan, ja tapahtuman ollessa tosi suorittaman jokin haluttu toimenpide. Tässä tapauksessa tapahtumaksi määriteltiin NETCONF-palvelimelta saapuva data ja toimenpiteeksi uuden tapahtuman luonti *New-Event* -cmdlet-ohjelmalla. Kun ohjelma toteaa, että NETCONF-palvelimen lähettämä viesti on saatu kokonaisuudessaan (palvelin lähettää merkkijonon *]]>]]>*), rekisteröi se uuden tapahtuman *New-Event*-komennolla, jonka *read-rpc*-komento lukee ja palauttaa käyttäjälle.

Käyttäjän datan lähetys NETCONF-palvelimelle toteutettiin *write-rpc*-komennolla [kuvan 23 kohta 4 (a)]. *Write-rpc*-komento kirjoitti XML-merkkijonon *connect-netconfsession*-funktiolla NETCONF-palvelimelle synkronisesti. Tämän jälkeen ohjelma kutsui *read*-rpc-funktiota, joka luki *New-Event*-funktion generoiman viestin Powershell-ympäristön viestijonosta.

Lopuksi lähetettiin NETCONF-istunnon lopetuspyyntö NETCONF-palvelimelle komennolla *disconnect-netconfsession*, kuvion 24 kohta 5 (a). Tämä tapahtui käytännössä siten, että NETCONF-palvelimelle lähetettiin merkkijono *<close-session/>*. NETCONF-palvelin

palautti tiedon siitä, että istunto saatiin päätettyä onnistuneesti tai se antoi virheilmoituksen jos, ilmeni ongelmia, kuten kuvion 24 kohdassa 5 (b) on esitetty.

5 Yhteenveto

Opinnäytetyössä ohjelmoitiin Powershell-skriptikielellä ohjelmistokirjasto NETCONF-protokollalle. Ainoastaan NETCONF-protokollan kuljetuskerroksen yhteys (SSH) oli toteutettava ulkoisella ohjelmalla (plink.exe). Powershell-ympäristön skriptikieli tuntuisikin tarjoavan työkalun, joka mahdollistaa samantapaisen ohjelmointiympäristön kuin muut skriptikielet, esimerkiksi Python tai Perl, mutta joka on erittäin vahvasti integroitu Windows-käyttöjärjestelmiä hyödyntäville alustoille.

Toisaalta, koska Powershell on suhteellisen uusi tuote, on sille tarjottava dokumentaatio huomattavasti suppeampaa kuin mitä muille kielille löytyy. Lisäksi ohjelmointia tukevia ympäristöjä eikä työkaluja vielä löydy kovin monta.

Opinnäytetyön toinen tutkittava osio, NETCONF-protokolla, on myös vasta taannoin standardoitu, ja osittain tästä syystä kovin moni valmistaja ei vielä tue sitä laitteissaan. Ongelmia tuottavat myös eri valmistajien toteutusten poikkeavuudet NETCONF-protokollan sisältökerrokselle. Tästä johtuen ei voida tehdä ohjelmistokirjastoa joka, sellaisenaan toimisi eri valmistajien tuotteilla, vaan suuri osa ohjelmoinnista on edelleen tehtävä valmistajakohtaisesti.

Kuitenkin voidaan katsoa, että Powershell-ympäristön skriptikieli tarjoaa toimivat työkalut ohjelmien prototyyppien kehittämiseen sekä ylläpidolle suunnattujen ohjelmien työstämiseen. Koska Microsoftin alkuperäinen käyttötarkoitus Powershell-ympäristölle oli automaation tarjoava alusta juurikin ylläpidollisiin toimenpiteisiin on odotettavissa, että myös tulevat versiot tulevat keskittymään ominaisuuksiltaan tämän kaltaisille ratkaisuille.

Opinnäytetyön tuloksena syntynyttä ohjelmaa tulisi kuitenkin kehittää tukemaan kaikkia NETCONF-protokollan määrittelemiä toimenpiteitä. Lisäksi ohjelma tulisi muuttaa siten, että se käyttäisi kuljetuskerroksen (SSH) muodostamiseen jotain .NET-

sovelluskehykseen pohjautuvaa avoimen lähdekoodin kirjastoa. Tämä tosin vaatisi myös kyseisten kirjastojen kehittämistä koska mikään nykyisistä kirjastoista ei tue NETCONF-protokollan vaatimaa SSH Subsystem -toimintoa.

Lähteet

1 Payette, Bruce. 2011. Powershell in Action. 2nd ed. Shelter Island: Manning Publications.

2 Snover, Jeffrey P. 2002. Verkkodokumentti. Monad Manifesto. Päivitetty 8.8.2002. Luettu 23.3.2012. <http://www.jsnover.com/Docs/MonadManifesto.pdf>.

3 Windows Powershell. 2012. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Windows_PowerShell>. Päivitetty 9.3.2012. Luettu 23.4.2012

4 How Windows Powershell Works. 2012. Verkkodokumentti. Microsoft. <http://msdn.microsoft.com/en-us/library/ms714658%28v=vs.85%29.aspx>. Luettu 13.4.2012.

5 PSHost Class. 2012. Verkkodokumentti. Microsoft. <http://msdn.microsoft.com/en-us/library/system.management.automation.host.pshost%28v=vs.85%29.aspx>. Luettu 13.4.2012.

6 Kumaravel, Arul; White, Jon; Li, Michael Naixin; Happell, Scott; Xie, George; Vutukuri, Krishna Chytanya; Chris Webb (toim.). 2008. Windows Powershell Programming: Snap-ins, Cmdlets, Hosts and Providers. Indianapolis : Wiley Publishing, Inc.

7 Mantl, Oliver. Verkkodokumentti. Threading in Powershell. <http://www.codeproject.com/Articles/261193/Threading-in-PowerShell>. Päivitetty 14.2.2012. Luettu 15. 4.2012.

8 Jacquenet, Christian;Bourdon, Gilles ja Boucadair, Mohamed. 2008. Service Automation and Dynamic Provisioning Techniques in IP/MPLS Environments. Chicester: John Wiley & Sons Ltd.

9 Network Configuration (netconf) - Charter. 2012. Verkkodokumentti. Internet Engineering Task Force. <http://datatracker.ietf.org/wg/netconf/charter/> Luettu 29.3.2012.

10 Netconf Central. 2012. Verkkodokumentti. <http://www.netconfcentral.org/>. Luettu 29.3.2012.

11 Enns, R. 2006. Verkkodokumentti. Internet Engineering Task Force.

12 Enns, Rob. 2011. Verkkodukumentti. RFC 6241: NETCONF Configuration Protocol. Internet Engineering Task Force.

13 OSI-malli. Verkkodokumentti. Wikipedia. http://fi.wikipedia.org/wiki/OSI-malli. Päivitetty 14.1.2012. Luettu 29.3.2012.

14 Stallings, William. Verkkodokumentti. The Internet Protocol Forum. <http://www.ipjforum.org/?p=180>. Päivitetty 2.2.2010. Luettu 10.4.2012.

15 Junos OS 12.1 NETCONF XML Management Protocol Guide. Verkkodokumentti. Juniper Networks. <http://www.juniper.net/techpubs/en_US/junos12.1/information-products/topic-collections/netconf-guide/netconf-guide.pdf>. Päivitetty 13.3.2012. Luettu 11.4.2012.

16 Bjorklund, M. 2010. RFC 6020: YANG - A Data Modeling Language for Network Configuration Protocol. Verkkodokumentti. Internet Engineering Task Force.

17 Ward, Brian. 2002. THE BOOK OF VMware. San Francisco: No Starch Press, Inc.

18 Cameron, Rob; Brad Woodberg; Patricio Giecco; Tim Eberhard; James Quinn. 2010. Junos Security. Sebastopol: O'Reilly Media.

19 Steenbergen, Richard. 2012. Verkkodokumentti. Juniper Clue. <http://juniper.cluepon.net/index.php/Olive>. Luettu 23.4.2012.

Junos-käyttöjärjestelmän konfiguraatio

```
## Last commit: 2012-03-24 17:01:33 UTC by jrapila
version 10.4R1.9;
system {
    host-name junos-netconf;
    domain-name joosus.local;
    root-authentication {
        encrypted-password "$1$HTMWz15N$z49MKMF9PeYYOmpaalzWI."; ## SECRET-
DATA
    }
    login {
        user jrapila {
            full-name "Joonas Rapila";
            uid 2000;
            class super-user;
            authentication {
                encrypted-password "$1$1BJI/Af5$BmIDZxeh/qVDF3CYGXZ4f/"; ## SECR
ET-DATA
            }
        }
        user netconf {
            uid 2001;
            class super-user;
            authentication {
                encrypted-password "$1$O9KWyMIH$2Ckpv1apRjx8p0VUXU1Bn1"; ##
SECR                                                                    ET-DATA
            }
        }
    }
    services {
        ssh {
            protocol-version v2;
```

```
        }
        netconf {
            ssh {
                connection-limit 50;
            }
        }
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file interactive-commands {
            interactive-commands any;
        }
    }
}
interfaces {
    em0 {
        unit 0 {
            family inet {
                address 10.0.1.249/24;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 127.0.0.1/32 {
                    preferred;
```

```
            }
        }
    }
}
routing-options {
    static {
        route 0.0.0.0/0 next-hop 10.0.1.1;
    }
}
```